MULTI-ROBOT FASTSLAM FOR LARGE DOMAINS

THESIS

Choyong Koperski, Captain, USAF

AFIT/GCE/ENG/07-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCE/ENG/07-06

MULTI-ROBOT FASTSLAM FOR LARGE DOMAINS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Choyong Koperski, BS

Captain, USAF

March 2007

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GCE/ENG/07-06

MULTI-ROBOT FASTSLAM FOR LARGE DOMAINS

Choyong Koperski, BS

Captain, USAF

Approved:

/SIGNED/

_____ _____

Gilbert Peterson, Ph.D. Chairman) Date

/SIGNED/

_____ _____

John Raquet, Ph.D. (Member) Date

/SIGNED/

_____ _____

Michael Veth, Major, USAF (Member) Date

AFIT/GCS/ENG/07-06

Abstract

For a robot to build a map of its surrounding area it must have accurate position information within the area and to obtain the accurate position information within the area, the robot need to have an accurate map of the area. This circular problem is the Simultaneous Localization and Mapping (SLAM) problem. An efficient algorithm to solve it is FastSLAM, which is based on the Rao-Blackwellized particle filter. FastSLAM solves the SLAM problem for single-robot mapping using particles to represent the posterior of the robot pose and the map. Each particle of the filter possesses its own global map which is likely to be a grid map. The memory space required for these entire maps pose a serious limitation to the algorithm's capability when the problem space is large. In addition this problem will only get worse if the algorithm is adapted to a multi-robot mapping. This thesis presents an alternate mapping algorithm that extends this single-robot FastSLAM algorithm to a multi-robot mapping algorithm that uses Absolute Space Representations to represent the world. However, each particle still maintains a local grid to map its vicinity and periodically this grid map is converted into an Absolute Space Representations. An Absolute Space Representation expresses a world in polygons requiring only minimal amount of memory space. By using this altered mapping strategy, the problem faced in FastSLAM when mapping a large domain can be alleviated. In this algorithm, each robot maps separately, and when two robots encounter each other they exchange range and odometry readings from their last encounter to this encounter. Each robot then sets up another filter for the other robots data and incrementally updates its

own map, incorporating the passed data and its own data at the same time. The passed data is processed in reverse by the receiving robot as if a virtual robot is back-tracking the path of the other robot. The algorithm is demonstrated using three data sets collected using a single robot equipped with odometry and laser-range finder sensors.

Table of Contents

Page

List of Figures

Page

List of Tables

Page

1. Introduction

1.1 Background

Building truly autonomous robots is a highly sought-after goal in the field of mobile robotics. Autonomous robots can be used for various purposes in our lives such as building maps for search and rescue operations and executing simple tasks in our offices and homes.

Like humans, robots rely on sensor measurements to interpret theirs environment and subsequently build maps using the data. Robots specifically use range-finders to read distances to structures in the environment. However, sensors including range-finders induce errors in readings due to sensor failure, sensor noise, environment noise, or inability to sense particular materials. In addition to the inaccuracies from sensors noise, robots carry another intrinsic limitation. Robot motion control systems are not totally reliable. In fact, the largest amount of error is often from the motion control system. In the midst of these limitations, researchers strive to develop truly autonomous robots. To have a reliably-functioning robot, it is important for the robot to have the capability to map its environment accurately. Without a map, a robot would not be able to navigate and perform simple tasks. However, to map an area accurately, the robot has to know its location within its environment. When its location is misinterpreted, subsequent sensor readings would be misinterpreted as well ending up building inaccurate maps. This induces a "chicken and egg" problem. To be able to map, the robot has to know its pose within the map, and, to know its position, it requires an accurate map. This nature gives

rise to the problem of Simultaneous Localization and Mapping (SLAM), known as SLAM.

SLAM is a fundamental capability required for navigating unknown environments when GPS data is not available. Robots without knowledge of their whereabouts can not perform given commands successfully. To be reliable, robots need capability to map their surroundings accurately in spite of errors caused by sensor readings and their motion control system. Many methods have been studied to explore the possibility of building efficient autonomous robots. Among them are the well known Extended Kalman Filter (EKF) and FastSLAM.

The Extended Kalman Filter (EKF) has been one of the most popular approaches to the SLAM problems for last two decades. Although this approach produces good results, it suffers from two limitations: quadratic complexity and sensitivity to failures in data association [64].

In recent years, an efficient and robust SLAM algorithm called FastSLAM has drawn significant attention from researchers in mobile robotics. This algorithm solves the SLAM problem for single-robot mapping by using the Rao-Blackwellized particle filter (RBPF) to approximate the posteriori estimate of a robot pose and the map. In the algorithm, each robot maintains a particle filter which is made of a number of samples. The samples estimate the posterior of the robots position and the possible maps. The mapping, often done on a grid, limits the number of samples which can be used in the algorithm, due to memory constraints. This work extends the original FastSLAM algorithm to alleviate this memory constraint by letting each sample maintain only a

small fine-grain local grid map and a global Absolute Space Representations (ASR) map best thought of as the polygons which represent the surfaces in an environment. This thesis also extends the single-robot FastSLAM algorithm to multi-robot mapping problems with unknown initial positions.

1.2 Problem Statement

Although FastSLAM [58] is a powerful solution to the SLAM problems, it suffers from a problem. The FastSLAM algorithm uses a particle filter to approximate the posterior of a robot given a motion model and builds a map adding robot observations incrementally using a Bayesian update step [58]. Per iteration, each particle of the path estimator estimates the posterior of the robot's pose and updates its map based on the estimated pose information. Then a new set of particles is extracted from the previous set of particles, based on the accuracy of the posterior and map estimations. Only particles with the best map, or highest posterior estimation, survive to the end and the best is selected to represent the learned map. This requires each particle to possess a global map.

When errors in the estimation of a robot's pose are great, a large number of particles are required to converge on a feasible map, and because each particle has its own map, the memory space required is quite large just to build a map for an area of $40000m^2$. This memory requirement limits the domain size of the FastSLAM algorithm. To overcome this shortcoming, this thesis introduces an online multi-robot FastSLAM algorithm in which each particle maintains a fine grained local grid map and a global map using Absolute Space Representations (ASRs). This thesis work shows that, by revising the original FastSLAM algorithm to this new map representation, the maximum number

of particles used for FastSLAM problems can be doubled without increasing the memory requirements.

1.3 Research Objectives

The objectives of this thesis are to:

1. Implement and test the single-robot FastSLAM algorithm using local grid maps and global Absolute Space Representations.
2. Implement and test multi-robot FastSLAM.
3. Develop a method to extract a polygonal map representation from a grid map and to simplify the extracted polylines.

The methodology associated with each of these objectives is discussed briefly in the following section.

1.4 Methodology

In multi-robot mapping, one can't assume that the robots will be accurately aware of their starting location and the starting locations of the other robots. Because of this, in multi-robot mapping, each robot initializes its own coordinate system and starts mapping as if from an unknown location. Robots don't know the locations of other robots and begin mapping until they encounters another robot. Upon encountering each other, the robots determine their relative poses to each other and exchange their collected range and odometry data from the time when they last met. Upon exchange, each robot sets up a virtual robot whose initial pose is set to the relative pose of the encountered robot(s). Then each of these virtual robots sets up a particle filter and begins mapping with the

received data in time-reversed order. The master robot and the virtual robot incorporate their partner's already developed map into their maps when passing through an overlapping area.

In order to reduce the amount of memory required for the particles, each particle of the master robot and the virtual robot periodically transform their local grid map into Absolute Space Representations (ASR) by extracting polygons from the grid map and simplifying the polygons, and the ASRs are stored until the robot's next visit to the area. When revisiting a previously mapped area and when starting the next local map, the overlapping area of the new local grid of each particle in the RBPF is initialized to the values of the previous local grid. When each robot returns to the previously visited area, it reopens the polygonal local map and incorporates the map data into its current map.

The algorithm is tested offline using three data sets obtained using a single robot equipped with odometry and laser-range finders. The first data set is processed in order and tested for the single-robot FastSLAM problem to demonstrate efficiency and effectiveness of the algorithm. In the second simulation, the second and the third data set are processed at the same time to simulate two robots encountering each other, but with a minimal amount of overlap. The third data set is collected from the ending location of the second data set. This means that there is a point where there is no difference in the actual poses of the robots. Results show that multi-robot mapping using FastSLAM and ASRs produces reliable maps with an error between the robots starting and ending locations of 0 cm.

1.5 Thesis Preview

Chapter 2 reviews related work done in robotic mapping, localization, and SLAM. This is followed by the proposed method for mult-robot FastSLAM, and in Chapter 4, implementation details and test results are shown and analyzed. The document concludes with conclusion about the results and potential future work.

2. Literature Review

This chapter presents related work in robotic mapping and Simultaneous Localization and Mapping (SLAM). First in Section 2.1, methods for performing single robot mapping are presented. Section 2.2 presents mathematical background material required for SLAM, and then in Section 2.3, research in localization is discussed. Finally, in Sections 2.4 and 2.5, SLAM and multi-robot SLAM are addressed.

2.1 Robot Mapping

The goal of the robot mapping problem is for the robot to determine the map of its environment, given sensor readings and the robot's pose/location. The mapping problem is analytically expressed as estimating the probability distribution of the map m_t,at time t given the previous times sensor s_{t-1} and pose z_{t-1} information and the previous map m_{t-1}, and is expressed as

$$p(m_t \mid s_{t-1}, z_{t-1}, m_{t-1}) \tag{2.1}$$

Since only the current and previous time slices are represented, the assumption is that the information from all previous time slices is encapsulated in the current state (the Markov assumption).

One of the problems in mapping is that the process is inherently noisy. Robots depend on sensor readings to map their environment. However, these readings are subject to errors. In addition, robot motors induce errors as well. When a move command is given to the robot with specified direction and distance, the robot performs the command and believes to be at the specified location, although the new location the robot ends up might be far from the believed-to-be location. The robot motors also lack the ability to

incorporate the state of the surface the robot is moving on, inducing more errors in its estimations. When these errors accumulate over time without being controlled, the robot fails to maintain its actual pose information and misinterprets sensor readings. The map generated by such a lost robot is far from the real environment. Therefore, the robot has to know its position accurately to generate quality map. This brings up a problem of solving the posterior of the robot's pose while generating the map. Because of these inherent problems, building accurate maps using mobile robots has been a difficult task.

Most of mapping algorithms use metric or occupancy grid maps and topological maps to represent the world.

2.1.1 Occupancy grid maps

Occupancy grid maps are first introduced by Moravec and Elfes in 1985 [31]. Moravec and Elfes represent the world as fine-grained grids where each grid cell is marked as occupied or free space. Occupancy grids represent an environment in a grid of "cells". Each cell maintains the probability that the cell is occupied by some object. A metric map displays the geometric features of the environment. The majority of the SLAM solutions uses occupancy grid maps and updates the maps using Bayesian methods when new data is obtained. Alternatively, Pagac, Nebot, and Durrant-Whyte [33] use the Dempster-Shafer belief functions instead of probabilities [31].

2.1.2 Topological maps

Topological maps, first introduced by Kuipers [31], show the connectivity of the environment, and in many cases discard most metric information. Topological maps

describe the environment as a graph with nodes and arcs. Nodes represent the spaces (rooms, hallways, etc.), and arcs represent the probability of transitions between the spaces. However, the distinction between topological mapping and metric mapping has been ambiguous due to the fact that topological maps often rely on geometric features for localization after the map is built.

There has been some work done on extracting topological maps from occupancy grid data [36], most topological mapping acquires topological information about the environment during the exploration process [32, 35]. For example, Rybski, et al. [34] build topological maps by identifying visual features in image data acquired during the exploration process, and Tovar, LaValle, and Murrieta [41] maintain a dynamically-updated visibility tree representation that captures the topology of the environment [31].

As a comparison, metric maps provide more information about the environment but require more storage and are sensitive to measurement errors. Although topological maps cannot represent the environment as concisely, they can represent much larger environments [31]. Most mapping techniques, whether metric or topological in nature, are probabilistic in some sense [31].

2.1.3 Related Work in Mapping

Maps are either world-centric or robot-centric [31]. World-centric maps maintain a map using a global reference system, and robot-centric maps are described in a local robot sensor based measurement space. Robot-centric maps are difficult to extrapolate from individual measurements to measurements at nearby, unexplored places. For this reason, world-centric maps have generally been preferred. Since the 1990s, probabilistic

approaches have dominated robotic mapping [6], and the robotic mapping problems solution of choice is some form of simultaneous robot localization [53, 54]. Maps usually describe the location of landmarks, or significant features in the environment. For a general overview of several well-known probabilistic mapping techniques see [38].

Incremental mapping algorithms are popular because they can be run in real time. A global map is built incrementally by converting the raw sensor data to a compact map composed of generalized polylines [4]. The global map is also represented by generalized polylines and merging the new polylines is done by merging similar line segments. An online, real-time incremental mapping method introduced in [52] uses likelihood maximization for robot pose and the map estimation. The algorithm maintains the map in a hierarchical fashion in which each robot maintains its own local map. Each robot communicates a subset of its scans to the central robot using corrected scan coordinates. The central mapper then integrates the scan information from the individual robots in real time.

The method introduced in [47] uses an incremental mapping technique. However, the algorithm represents the two-dimensional map using manifolds. A manifold is an arbitrarily complex structure with varying local curvature and is discretized by dividing it into a set of overlapping patches. The algorithm takes maximum likelihood estimation techniques and adapts them for the manifold representation. The algorithm maintains a set of relations that contain the patches' relative pose. Each patch has finite extent and defines a local coordinate system [47]. A same location may be represented more than once in the manifold set.

Some researchers have worked on using both the metric and topological mapping approaches by maintaining local metric information (i.e. occupancy data) at topological nodes [37]. Tomatis, Nourbakhsh, and Siegwart [39, 40] represent hallways using topological maps and rooms using metric maps [31]. Laviers[67] initially uses occupancy grid maps to map rooms and then transforms the map into absolute space representations in which the world is represented as a series of connected spaces.

Although the majority of algorithms which perform mapping employ range sensors, some work is done using vision [53, 63]. Davison et al. use cameras that have the ability to fixate and to change fixation over a wide angular range. Using this method, re-detected features after a long time can be re-matched. Measurement of a feature in the map involves the stereo head using a prediction scheme to turn to fixate to the feature.

The following section discusses foundational material required for the discussion of mapping algorithm specifics. In Section 2.2.1, Bayes filtering and its association with Simultaneous Localization and Mapping (SLAM) are discussed. Kalman filters and particle filters are discussed in Sections 2.2.2 and 2.2.3 respectively. Finally, the Expectation Maximization algorithm is presented in 2.2.4.

2.2 Background Material

This section introduces several topics that are built on during the discussion of localization and mapping research in the remainder of the chapter and in Chapter 3. The first topic is Bayes Filter.

2.2.1 Bayes Filter [45]

Most of the localization, mapping, and SLAM approaches use probabilistic algorithms and make use of Bayes rule [28] which is

$$P(x|d)\,P(d) = P(x,d). \tag{2.2}$$

where $P(x|d)$ is the conditioned probability of x conditioned on d, $P(d)$ is the probability of d, and $P(x,d)$ is the joint probability of x and d.

By symmetry we can also

$$P(d|x)\,P(x) = P(x,d). \tag{2.3}$$

where $P(d|x)$ is the conditioned probability of x conditioned on x and $P(x)$ is the probability of x.

When only terms on the left hand side are rearranged;

$$P(x|d)\,P(d) = P(d|x)\,P(x). \tag{2.4}$$

From this, we can get

$$P(x|d) = \eta\,P(d|x)\,P(x) \tag{2.5}$$

when $\eta = 1/P(d)$. Here, x is the unknown state of the dynamic discrete system, d is measurement data and η is the normalizing factor. The term $P(d|x)$ represents the probability of observing the measurement d given x. The term $P(x)$ specifies the probability of the prior world being in the state of x. According to the Bayes' rule, the unknown data can be obtained by multiplying the probability of measurement data given the state x and the probability of the state being in x.

The Bayes filter extends Bayes' rule to temporal estimation problems [1]. It recursively estimates posterior probability distributions over quantities that cannot be

observed directly. Let x_t denote for the unknown quantity at time t, then x_t represents the state at time t and a posteriori probability over the state x_t can be calculated via the following recursive equation:

$$p(x_t|\, z^t,u^t) = \eta\, p(z_t|x_t) \int p(x_t|u_t,x_{t-1})p(x_{t-1}|z^{t-1},u^{t-1})\mathrm{d}x_{t-1} \qquad (2.6)$$

Here a superscript t refers to all data leading up to time t, that is:

$$z^t = \P z_1,z_2,z_3,\ldots,z_t\diamondsuit \qquad (2.7)$$

$$u^t = \P u_1,u_2,u_3,\ldots,u_t\diamondsuit \qquad (2.8)$$

As can be seen the equation (2.6) is recursive. The posterior probability $p(x_t|z^t,u^t)$ is calculated from the previous probability back to t=0, $p(x_0\,|z^0,u^0) = p(x_0)$. This recursive nature is the Markov assumption and this assumption enables Bayes filters to integrate information indefinitely. The Markov assumption assumes that the values in any state are only influenced by the values of the state and effects that directly precede it.

In Bayes filters, the state x_t contains all unknown quantities that may influence sensor measurements at multiple points in time. In robotic localization and mapping, unknown quantities are usually the location the robot is in and the map of the environment. Since both quantities influence sensor measurements in the future states, they have to be estimated together.

Hence, by denoting the map as m and the robot's pose as s, then x_t in equation (2.6) can be replaced with s_t and m_t and the equation can be rewritten as:

$$p(s_t,m_t|z^t,u^t) = \eta\, p(z_t|s_t,m_t)\int\int p(s_t,m_t|\, u_t,s_{t-1},m_{t-1}\,)\; p(s_{t-1},m_{t-1}\,|z_{t-1},u_{t-1})\; \mathrm{d}s_{t-1}\,\mathrm{d}m_{t-1} \qquad (2.9)$$

However, if the world is assumed to be static, then the time index for the map can be omitted. In addition, if we assume that the robot motion is independent of the map, this results in:

$$p(s_t,m|z^t,u^t) = \eta\, p(z_t|s_t,m) \int p(s_t|u_t,s_{t-1})p(s_{t-1},m|z^{t-1},u^{t-1})\, ds_{t-1} \qquad (2.10)$$

This estimator does not require integration over maps m, as was the case for the equation (2.9). Since a map usually involves a high dimensionality of the state, this problem reduction enables the problem of mapping and localization to be a sizable one.

We notice that to calculate $p(s_t,m|z^t,u^t)$, we need two distributions: $p(z_t|s_t,m)$ and $p(s_t|u_t,s_{t-1})$. However, these distributions are commonly assumed not to depend on the time t. Then they can be written as $p(z|s,m)$ and $p(s|u,s')$. In robotic localization and mapping, the probability $p(z|s,m)$ represents the perceptual model that describes the workings of the robot's sensors. The probability $p(s|u,s')$ represents the motion model of the robot. It models the effect of the robot controls on the state s'. Equation (2.10) is exact and can be used for any system for which the Markov assumption holds.

Many different probabilistic algorithms have been developed to approximate the equation for robotic mapping. Three popular approaches are based on the Kalman Filter (KF), Particle Filters (PF), and Expectation Maximization algorithm (EM).

2.2.2 Kalman Filter [22]

The KF is a recursive filter which estimates the state of a dynamic system with a series of incomplete, noisy measurements with an assumption that the system is linear and the noise is Gaussian. Under this assumption, the system is described as:

$$x_{t+1}=F_t\, x_t + w_t \qquad (2.11)$$

$$x_t z_{t+1} = H_t z_t + v_t \tag{2.12}$$

where w_t and v_t are the system observation noises and are zero-mean Gaussian distributions. F_t is the state transition model which is applied to the previous state x_t and H_t is the observation model which maps the true state space into the observed space. The KF represents the distribution by their means and covariance matrix. The update of the KF alternates between two stages: predict and update. In the predict stage, the predicted next state x'_{t+1} and predicted estimate covariance Σ'_{t+1} are calculated as

$$x'_{t+1} = F_t x_t \tag{2.13}$$

$$\Sigma'_{t+1} = F_t \Sigma'_t F_t^{\mathrm{T}} + w_t . \tag{2.14}$$

where the subscript letters t and $t+1$ indicate time and the superscript letter T means the transpose of a vector or matrix.

During the update stage, an optimal Kalman gain K_{t+1}, updated state estimate x_{t+1}, and updated estimate covariance Σ_{t+1} are calculated as:

$$K_{t+1} = \Sigma'_{t+1} H_t (H_t \Sigma'_{t+1} H_t^{\mathrm{T}} + \Sigma_t v_t)^{-1} \tag{2.15}$$

$$x_{t+1} = x'_{t+1} + K_{t+1}(z_t - H_t x'_{t+1}) \tag{2.16}$$

$$\Sigma_{t+1} = (I - K_{t+1} H_t) \Sigma'_{t+1} \tag{2.17}$$

However, it is often the case that the system is nonlinear and the Gaussian noise assumption for the motion and observation models is not adequate for such systems. To overcome the problem, the Extended Kalman Filter (EKF) [56] uses the KF with non-linear models.

2.2.3 Particle Filter

Instead of attempting to represent an entire probability distribution, a particle filter estimates the probability distribution using a large set of particles. Each particle represents a single belief of the system and carries an importance weight. As new information comes into the system, the particle filters update over three steps: sampling, importance weighting, and resampling.

Initially a certain number of particles are uniformly spread over the region of interest. At each time step, as new data arrives, each particle estimates the next state of the system. Once each particle has updated their belief, a new weight is calculated for each particle based on its new estimation. During the resampling stage, a new generation of samples is drawn from the current set of samples using the importance weight to weight sample selection. Samples with higher weight are likely to survive and have numerous descendents, while samples with lower weights are likely to die off. The number of particles required for a good approximation of a Bayesian system grows exponentially with the dimension of the state to estimate. To overcome this problem, Rao-Blackwellized particle filters (RBPF) are used, which make use of d-separation or Bayesian independence between state variables. RBPF is discussed more with the description of FastSlam in Section 2.5.

2.2.4 Expectation Maximization Algorithm [92]

The EM algorithm is an iterative procedure to compute the Maximum Likelihood (ML) estimate in the presence of missing data. It iterates between two processes: The E-

step and the M-step. For the E-step, an expectation of the likelihood of observation of unobservable data is computed. This is followed by finding the maximum likelihood estimates of the parameters using the expected likelihoods during the M-step [2]. Let y denote incomplete data and x the missing data. The x can either be actual missing measurement or a hidden variable that would make the problem easier if its value were known. Where $p(x,y|\ q)$ is the joint probability distribution of the complete data, given q, a parameters vector. Then, using the Bayes' rule and law of total probability, the distribution of the missing data is:

$$p(x \mid y,q) = \frac{p(y,x \mid q)}{p(y \mid q)} = \frac{p(y \mid x,q)\,p(x \mid q)}{\int p(y \mid x,q)\,p(x \mid q)dx} \tag{2.18}$$

This formulation requires two distributions: $p(y|x,\ q)$ and $p(x|q)$. The distribution $p(y|x,q)$ is the observation likelihood given the unobservable data, and the distribution $p(x|q)$ is the probability of the unobservable data. An EM algorithm iteratively improves an initial estimate q_0 alternating between the E-step and the M-step. q_{t+1} is estimated from q_t and obtained by maximizing the conditional expectation log likelihood given the observed variables under the previous parameter value, which is described as:

$$q_{t+1} = \text{argmax}_q\, E_x[\log p(y\ x|q_t)|\ y\ ] \tag{2.19}$$

Here E_x denotes the conditional expectation of log-likelihood, $log\ p(y,x|q)$. Log-likelihood is used instead of true likelihood $p(y,x|q)$, because it attains the same maximum point as the true likelihood while the problem becomes much easier to solve.

2.3 Localization

Section 2.1 introduced the topic of mapping, including a brief discussion of some of the techniques involved. The discussion notes that one of the biggest difficulties associated with mapping is sensor noise and error. If the robot had perfect pose information, then the mapping problem becomes almost trivial. Because of that, the following section discusses techniques that perform localization which improves the robots pose.

The problem of localizing a mobile robot is to determine the robot's pose in an environment, given the observation history, the command history, and a map of the environment. This problem is analytically expressed as estimating the probability distribution:

$$p(s_t|z_{0:t},u_{0:t},m) \tag{2.20}$$

Here s_t is the robot position to be estimated at time t, $z_{0:t}=\{z_0, z_1,\ldots,z_t\}$ and $u_{0:t}=\{u_0, u_1,\ldots,u_t\}$ are the sensor measurement and the motion command histories up to time t, respectively, and m is the map of the environment.

One simple robot localization method is robot pose tracking [7]. In this approach, the initial robot pose is known, and the consequent robot pose is estimated by compensating errors incrementally in a robot's odometry. Algorithms for tracking the robot poses in such a manner often make restrictive assumptions on the size of the error and the shape of the robot's uncertainty. However, the robot localization problem becomes more challenging when the robot's initial pose is not known a priori, which is called the global localization problem [8, 13]. In the global localization problem, the error

in the robot's estimate cannot be assumed to be small, hence, it requires a robot be able to handle multiple, distinct hypotheses. Even more challenging is the kidnapped robot problem [5] in which a well-localized robot is picked up and transported to an unknown place without being told. The vast majority of existing algorithms address only the position tracking problem, in which errors are assumed to be small and incremental. This property makes algorithms such as Kalman filters [12, 14] applicable in position tracking.

The position tracking approach calculates the robot's new pose based on the previous position and can be expressed as an estimation of the probability distribution:

$$p(s_t|z_{t-1},u_{t-1},s_{t-1},m) \tag{2.21}$$

Here s_t is the robot position to be estimated at time t and m is the static map. The variables z_{t-1}, u_{t-1}, and s_{t-1} denote the sensor measurement, motion command, and the robot pose at time $t-1$ respectively. In position tracking, the robot pose is usually represented using a Gaussian distribution, tracking only a single mode of the posterior. It is often unlikely to recover to a correct solution from an incorrect estimation.

On the other hand, the latter approach maintains multi-modes of the robot posterior as expressed by equation (2.20). This approach can deal with environment ambiguities and large robot displacements at the expense of an increased computational complexity.

Most robot localization solutions use sensors such as sonar and laser range-finders to localize the robot inside the map. The robot detects landmarks and localizes itself relative to landmarks, either using topological [15, 16] or grid-based maps [33, 44, 48]. Metric representations of the space [9, 8, 39, 45] are advantageous over topological

because these allow a variety of environments to be dealt with and are not restricted to orthogonal environments containing pre-defined or easily distinguishable features such as corridors, intersections, and doors. For topological localization, a few algorithms use artificial landmarks such as reflecting tape, ultrasonic beacons, buoys or visual patterns that are easily recognized to localize the robot. Some approaches use more natural landmarks such as hallways, openings, doors, walls, ceiling lights [17], and other vertical objects [18].

Although some approaches directly incorporates raw sensor data [27, 21, 14] directly into the maps, the majority of recent localization algorithms are probabilistic and use the history of the data read from sensors to estimate a position that is updated upon the arrival of new sensor readings [25]. Of these, although range finding sensors are most common, there are also vision-based localization approaches [53], as well as system which use wireless hubs to update the robots pose [25, 26].

Three representative approaches for robot localization are Kalman Filter, Markov, and Monte Carle localizations.

2.3.1 Kalman Filter Localization

Kalman filters estimate posterior distributions of robot poses conditioned on sensor data and represent posteriors by Gaussians. A localization problem of a KF can be described as:

$$s_{t+1}=A_t\, s_t+w_t \tag{2.22}$$

$$x_t z_{t+1}=H_t\, z_t+v_t \tag{2.23}$$

Here A_t is the action model which is applied to the previous robot pose s_t and H_t is the sensor observation model. The variables w_t and v_t are the action and observation noises respectively and are zero-mean Gaussian distributions, and z_t is the observation made by the robot at time t. In the predict stage, predicted next robot pose s'_{t+1} and predicted estimate covarianceΣ'_{t+1} is calculated as:

$$s'_{t+1}=A_t\, s_t \tag{2.24}$$

$$\Sigma'_{t+1}=A_t\, \Sigma'_t A_t^{\mathrm{T}} + w_t \tag{2.25}$$

where the subscript letters t and $t+1$ indicate time and the superscript letter T means the transpose of a vector or matrix. Then during the update stage, an optimal Kalman gain K_{t+1}, updated state estimate s_{t+1}, and updated estimate covariance Σ_{t+1} are calculated as:

$$K_{t+1} = \Sigma'_{t+1}\, H_t (H_t \Sigma'_{t+1}\, H_t^{\mathrm{T}} + \Sigma v_t)^{-1} \tag{2.26}$$

$$s_{t+1} = s'_{t+1} + K_{t+1}(z_t - H_t\, s'_{t+1}) \tag{2.27}$$

$$\Sigma_{t+1} = (\mathrm{I} - K_{t+1}\, H_t)\, \Sigma'_{t+1} \tag{2.28}$$

Although Kalman filters offer an elegant and efficient algorithm for localization, they make restrictive Gaussian noise and Gaussian-distributed initial uncertainty assumptions. Many algorithms have been developed to overcome this limitation. One approach uses multi-hypothesis Kalman filters. Multi-hypothesis Kalman filters represent beliefs using multi-modal Gaussians [10, 13], enabling them to pursue multiple, distinct hypotheses. However, this approach inherits from Kalman filters the Gaussian noise assumption. Another method to handle the globalization problem is Markov localization.

2.3.2 Markov Localization

Markov localization algorithms assume that the environment is static, and the robot's location is the only state in the environment which systematically affects sensor readings. Markov localization estimates the posterior distribution over s_t given all available data. This localization problem is described as

$$p(s_t=S_t|d) \tag{2.29}$$

where s_t is the estimate of the current robot pose, S_t is the true robot pose at time t, and d is the collection of all the data available up to time t-1. Markov localization algorithms require all positions to be updated after each sensor reading and fail to localize a robot if too many aspects of the environment are not covered by the world model. Some Markov localization algorithms use features [15] or raw sensor data [9, 5]. However, using sensor data requires fine-grain grid representations of the environment that requires maintaining a huge state space. To overcome this problem, alternate algorithms using selective updating algorithms [5] and a tree-based representation [8] have been proposed. Burgard et. al [5] extends Markov localization for dynamic environment. The algorithm uses *entropy* or *distance filters* and a fine-grained grid representation of the robot's belief state. The algorithm updates the position probability density using only those measurements which are with high likelihood produced by known objects contained in the map [5]. The algorithm does not depend on abstract features, enabling raw sensor data to be incorporated into the robot's belief. Distance filters have the advantage that they do not average over all possible situations and that their decision is based on the current belief of the robot. Besides, entropy filter has an advantage of making no assumptions

about the nature of the sensor data and the kind of disturbances occurring in dynamic environments. Another popular localization approach is Monte Carlo Localization (MCL).

2.3.3 Monte Carlo Localization

Monte Carlo Localization (MCL) is a recursive Bayes filter that estimates the posterior distribution of robot poses as conditioned by the sensor data. MCL assumes the Markov assumption and the posterior of the robot pose is described as

$$\mathrm{Bel}(s_t)=\mathrm{p}(s_t|z_t,z_{t-1},...,z_0,u_t,u_{t-1},\ldots,u_0) \tag{2.30}$$

where s_t is the pose of the robot, z_t the sensor data, and u_t is the robot motion command at time t. The word "Bel' stands for the robot's belief in the robot pose at time t.

A MCL algorithm, Sequential Monte Carlo Methods (SMC) [61], uses two main ideas, sequential importance sampling and resampling, to update the posterior distribution as observation data become available which arrive sequentially. At each time instance t, SMC generates a collection of N weighted random particles [22], according to the importance density. The paths until time t-1 are not modified. In the resampling stage, particles with larger weights are sampled multiple times, while particles with small weights die away. Then an equal weight is assigned to each particle of a new sample set of particles is assigned.

2.3.4 Multi-Robot Localization

Most work with emphasis on robot localization is performed using a single robot. However, robot localization using cooperative robots to reduce the odometry error have also been studied.

Kurazume and Shigemi [19], for example, use two groups of robots. When the robots in one group move the other group members are required to remain at their positions. After each a motion command, all robots stop, perceive their relative position, and use this to reduce errors in odometry. Rekleitis and colleagues [20] present a similar method, in which a robot remains stationary while another robot moves and tracks the position of the moving robot. However the two mentioned methods lack capability to solve the global localization problem.

On the other hand, robots in [27] individually localize themselves using a sample-based version of Markov localization. Each robot maintains its own belief state, and the estimation of the posteriors is carried out locally on each robot. Individual robots perform Monte Carlo Localization that represents the beliefs of its own state using a set of weighted random samples, or particles. When a robot detects another robot using cameras and range-finders, they synchronize their belief states. The method approximates the robot belief states using piecewise constant density functions represented by a tree. When the robots establish correspondence between two belief states, the density values of the tree are multiplied into the other robot's sample set. This method helps the robots localize themselves faster with higher accuracy, even without initial knowledge of their location.

In this method, only "positive" detections are allowed and robots can't see each other at the same time.

2.4 Simultaneous Localization and Mapping (SLAM)

Robots require sensor and odometry readings to map its environment. However, sensors used for environment detection and odometry readings have inherent errors, which induces the SLAM problem. SLAM addresses building a map of an unknown environment while the robot also tries to localize itself inside the map, and it is summarized as estimating the probability distribution:

$$p(s_t, m_t | z_{0:t}, u_{0:t}) \tag{2.31}$$

A main issue in the SLAM problems is to decide how to represent the joint distribution over robot poses and maps. Of the state-of-the-art probabilistic approaches to the SLAM problem are Kalman filters, Expectation Maximization (EM) method, and fastSLAM.

2.4.1 Kalman Filter SLAM

One popular probabilistic approach to SLAM uses Kalman filters [1]. For mapping the Kalman filters represent posterior estimates $p(s_t, m | z^t, u^t)$ with Gaussians [1]. In the SLAM problem, the state x is a vector that comprises the robot's pose s and the map m [1]:

$$x_t = (s_t, m)^{\mathrm{T}} \tag{2.32}$$

From this point on, T means the transpose of a vector or matrix. In 2-dimentional Cartesian coordinates, the robot pose s is broken down into three variables s_x, s_y, and θ. s_x

and s_y are the robot poses in the x and y coordinates and θ represents the robot heading. Maps in the Kalman filter approach maintain the Cartesian coordinates of sets of features [1]. Features may include landmarks, objects or special shapes in the environment. Since there can be a number of features in the environment, the length of the vector x_t can become large. If K represents the total number of unknown features, then x_t becomes a $2K$+3-dimensional vector:

$$x_t = (s_{x,t}, s_{y,t}, \theta_t, m_{1,x,t}, m_{1,y,t}, m_{2,x,t}, m_{2,y,t}, \ldots, m_{k,x,t}, m_{k,y,t})^T \qquad (2.33)$$

Here $m_{k,x,t}$ and $m_{k,y,t}$ are Cartesian coordinates of the k-th feature in the map. The posterior over the robot pose and landmarks estimate $p(s_t, m | z^t, u^t)$ is described by a mean vector μ_t and a covariance matrix Σ_t. Here, the mean vector has $2K$+3 dimensions, and the covariance $(2K+3)^2$ dimensions.

Kalman filter SLAM approaches require three basic assumptions. First, the next state function must be linear with added Gaussian noise. Second, the perceptual model must also be linear with added Gaussian sensor noises. And third, the initial uncertainty must be Gaussian [1].

Using the linear next state function, the post s_t and the map m_t depends linearly on the previous pose, s_{t-1}, the previous map, m_{t-1}, and the control u_t. Since the environment is static, approximating the robot pose and the map by a linear next state function is appropriate. However, in reality robot poses is usually governed by a nonlinear trigonometric function that depends nonlinearly on the previous pose and the robot controls [1]. Matrix multiplications are the most costly operations in updating the Kalman

filter. Kalman filter-based algorithms require time quadratic in the number of landmarks to incorporate each sensor observation [58].

To date, the Kalman filter approach is the only technique that can estimate the full posterior over maps and robot poses in an online fashion. However, the assumption that measurement noise must be independent and Gaussian poses a limitation for practical implementations when features are indistinguishable. In addition, it doesn't handle correspondence problems well. The correspondence problem associates sensor measurements taken over time to each other. For these limitations, an environment with a sparse set of features that can be reliably identified can be best mapped using Kalman filters. Paskin [66] implements the KF approach using a thin junction tree. The junction tree itself represents the actual belief state. The tree expresses correlations as edges for direct dependencies and as paths for indirect dependencies. Each filter update adds an edge to the tree.

2.4.1.1 Extended Information Filters

The Extended Kalman Filter extends the standard the KF by incorporating non-linearity in the models. In the extended Kalman filter, single motion commands are broken into a series of smaller motion segments to account for nonlinearity. The EKF incrementally estimates the joint posterior distribution over the robot and landmark positions [64], and is the most popular approach to the SLAM problem. However, EKF-based SLAM approaches still share the same problem in time complexity with the KF which takes time quadratic to update the number of landmarks, because they maintain correlations between landmarks. This limits the application of the approach to the

environment in which only a few hundred features are contained. In addition, this approach assumes that landmarks are identifiable, which is not often true in the real world. In [53], the robot performs a stereo image search of size determined by the innovation covariance and uses the head's known odometry and its matched image coordinates to produce a measurement of the feature position relative to the robot. To reduce the computation, his method performs selective sensing. Since the work was not done within a multiple hypothesis framework with linear models, a single mismatch can be fatal to the localization process and, when uncertainty in the map is large, EKF updates is unpredictable due to the unmodelled non-linearity in the system.

The SLAM method in [24] uses an EKF using a postponement technique. It maintains a single map with a submap. It uses a submap to postpone the full map update. A submap is the set of features most recently observed, and as a new feature is observed the submap is expanded to include the new feature. In this way, the full map update can be postponed and the time complexity is improved by an order of magnitude. A compressed filter in [29] is identical to the postponement technique of [24]. The only difference lies in the set of the maintained features that are defined by geographical boundaries rather than expanded dynamically.

2.4.1.2 Sparse Extended Information Filters

Sparse extended information filters (SEIF) [79] represents the robot and landmarks posterior by a sparse Gaussian Markov random field. SEIF is similar to the thin junction tree method in [66] and represents maps through local, Web-like networks of features. SEIF is based on the information form of the EKF, known as the extended

information filter (EIF), and is motivated by the observation that the inverse covariance matrix of the EKF is sparse in nature. The elements of the matrix are treated as links between the locations of different features. The stronger the link is, the closer the landmarks. SEIF maintains a sparse information filter, in which nearby features are linked through a non-zero element. At each update, edges are removed to guarantee a constant-filter operation. SEIF updates the single, global map in constant time per step, except during loop closing.

2.4.2 Expectation Maximization SLAM Algorithms

In using the EM algorithm for SLAM, the function that is being maximized is the expectation over the joint log likelihood of the data d^t and the robot path $s^t = \{s_1,\ldots,s_t\}$ [1]:

$$m^{[i+1]} = \text{argmax}_m E_{st}[\log p(d^t,s^t|m)|m^{[i]},d^t] \tag{2.34}$$

where E_{st} denotes the conditional expectation of $\log p(d^t,s^t|m)$ given the previous map, a sensor reading, and a motion command

With Markov assumptions, equation (2.34) can be re-expressed as

$$m^{[i+1]} = \text{argmax}_m \sum_{\tau} \int p(s_\tau|m^{[i]},d^t) \log p(z_\tau|s_\tau,m) \, d\, s_\tau \tag{2.35}$$

The term $p(s_\tau|m^{[i]}\, d^t)$ is the posterior estimate for the robot pose s_τ given the data d_t and the i-th map $m^{[i]}$.

The robot posterior estimate $p(s_\tau|m^{[i]},d^t)$ for a given map is calculated in the E-step, and the most likely map for all poses s_t is obtained during the M-step from the previous map by fixing the expectations $p(s_\tau|m^{[i]}\, d^t)$ and by maximizing the log likelihood of the sensor measurements $\log p(z_\tau|s_\tau,m)$ [1]. At each iteration step, more accurate maps

are calculated. An empty map is used as the initial map. Since the robot's path s^t is unknown, equation (2.34) computes the expectation of likelihood over all possible paths the robot may have taken [1].

EM algorithms can solve the correspondence problem by repeatedly updating the pose posterior in the E-step. Since the obtained posterior means a different hypothesis on the robot's path, it implies different correspondence and these correspondences become features in the map obtained in the M-step [1]. However, EM algorithms can not retain a full notion of uncertainty, suffering from local maxima. In addition, They can't generate maps incrementally.

The SLAM algorithm in [66] performs efficient maximum likelihood projections periodically to remove weak or redundant edges. The junction tree gives immediate access to the marginal distribution over any variable, thus enables the process of selecting an edge to prune and pruning them in a linear-time filtering operation. The method also delays the incorporation of recent evidence into the map.

2.4.3 FastSLAM

Another popular approach to solving the SLAM problems is FastSLAM. The FastSLAM algorithm [58] approaches the SLAM problem from a Bayesian point of view. Figure 2.1 shows the single-robot SLAM problem as a dynamic Bayes network.

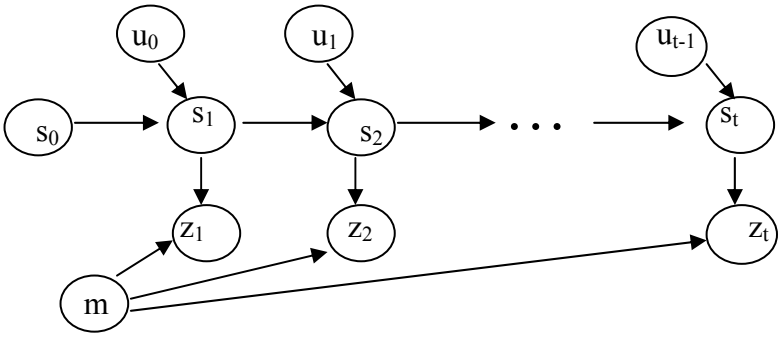


Figure 2.1: Bayes Net for single-robot SLAM problem.

Here s_i stands for a robot pose, z_i an observation made by the robot, and u_i an robot motor command given at time i. Following the notation of [69], $s_{1:t}$ denotes a sequence of robot poses given up to time t, $z_{1:t}$ a sequence of observations, and $u_{1:t}$ a sequence of robot actions. The m stands for a map. The FastSLAM algorithm utilizes the fact that the map and the path are independent via d-separation. This property was first observed by Murphy [59]. Based on this observation, Montemerlo, et al. decomposes the SLAM problem into the robot posterior and map estimation problems [58].

The algorithm makes use of a Rao-Blackwellized particle filter in which each particle of the filter maintains an estimate of the robot posterior and a map estimate. Particles evolve over time by sampling a new set of particles from the old set of particles based on an importance factor. The importance factor is calculated based on the current map, a robot motion command, and current sensor measurements. Sensor updates in the FastSLAM algorithm are done by a simple ray-tracing. The complexity of the Rao-Backwellized particle filter algorithm is O(MK) when we reduce a landmark to a point in

the plane. Here, M is the number of particles and K is the number of landmarks. Single-robot SLAM can be described as the problem of estimating $P(s_{1:t}, m| z_{1:t}, u_{0:t-1}, s_0)$. Utilizing Murphy's observation, this problem can be restated as

$$p(s_{1:t},m|z_{1:t},u_{0:t-1},s_0)= p(m|s_{1:t},z_{1:t},u_{0:t-1},s_0)p(s_{1:t}|z_{1:t},u_{0:t-1},s_0) \quad (2.36)$$

The first term is a distribution over possible maps and the second is a distribution over possible robot's posterior pose. As is shown, the first term depends on the second term, and a particle estimates the second term using a motion model. Since each particle maintains its own pose estimate, it also builds a separate map. If we let $s_t^{(i)}$denote for the pose, $m_t^{(i)}$ for the map, and $w_t^{(i)}$ the importance weight of i_{th} sample at time t, the filter's update function is summarized as follows:

$$s_t^{(i)}= A(u_{t-1},s_{t-1}^{(i)},z_t) \quad (2.37)$$

$$m_t^{(i)}= M(s_t^{(i)},z_t) + m_{t-1}^{(i)} \quad (2.38)$$

$$w_t^{(i)}= S(z_t,s_t^{(i)},m_{t-1}^{(i)})w_{t-1}^{(i)} \quad (2.39)$$

where A, M, and S are the action, map, and sensor models, respectively. An action model randomly selects a pose from the distribution of the first term in (2.36). The action model depends on only robot motion commands, induces a large state space can require a large number of particles. To overcome this limitation, more accurate action models can be used by incorporating observation data. The weights of the particles with self-consistent maps will be larger and survive to the next time step, while those of particles with inconsistent maps are assigned smaller values which cause the sample to be removed from further evolution. Hu, et al. [63] use a modified particle filter that maintains separate particles to represent the landmarks and RBPF is used to estimate a robot path posterior.

This modified particle filter nonlinearly approximates the robot path posterior and maps at every point in time. The next section presents DP-SLAM that is a transformation of the FastSLAM algorithm

2.4.3.1 DP-SLAM

DP-SLAM [65] uses a distributed particle mapping to build a map in real-time despite imperfect information about the robot's trajectory in the environment. The approach exploits the conditional independences noted by Murphy. However, the approach is laser based and makes no landmark assumptions which are often the case for most of the approaches to SLAM. The approach avoids data association problem by storing multiple detailed maps instead of sparse landmarks. The DP-SLAM algorithm uses a particle filter to represent both robot poses and the map and a particle ancestry tree. The tree itself is rooted with an initial particle, and each particle maintains a pointer to its parent and is assigned a unique numerical ID. When a particle is sampled to produce a successor particle, the successor is given an ID and a pointer to the predecessor particle. The tree has a bounded size which is maintained by pruning away unnecessary nodes. DP-mapping maintains a single occupancy grid, where each grid square stores a balanced tree. Each particle maintains a list of grid squares that it has updated [65]. The worst time complexity for incorporating a laser sweep into the map is log-quadratic in the number of particles and linear in the area swept out by the laser. So far single-robot mapping methods have been presented. In the following section, multi-robot mapping techniques are presented.

2.5 Multi-Robot SLAM

Although most research on SLAM is done using a single robot, some research has been conducted on using multiple robots for the SLAM problem. Most of the work done on multi-robot SLAM is done under certain constraints about the robots' initial pose. The robots' initial pose is known exactly [51, 52] or is approximated [68, 11]. However while other algorithms assume that initial pose relative to each other is unknown. The algorithm in [62] localizes robots in each other's maps using particle filters. Others compare the local maps [30], or set a correspondence list to determine rotation and translation amount. Maps can even change shape in the map fusion process [30, 62]. Cooperative mapping and localization is addressed in [57, 60].

Sparse extended information filters (SEIF) [30] extends [56] to the multi-robot SLAM with techniques for establishing correspondence between maps gathered by multiple robots. Uncertainties in map building is handled by using a sparse information filter technique, which represents maps and robot poses representing the maps by sparse Gaussian Markov random fields(GMRFs). This method builds maps even when relative starting location is unknown and landmarks are ambiguous. Each robot maintains its own local map and posterior in its own local coordinate system. All updates are additive and done locally. Each robot updates the map confined to its own pose and landmarks previously detected by it. Map fusion between two robots is performed by finding a relative coordinate transformation between the two maps (translation and rotation), and a correspondence list of landmarks. It aligns the two maps to obtain a single map by a tree-based algorithm for searching similar-looking local landmark configurations, paired with

a hill climbing algorithm that maximizes the overall likelihood by search in the space of correspondence.

Williams [55] extends the Constrained Local Submap Filter (CLSF) for the single-robot SLAM algorithm [54] to the multi-robot SLAM problems. It maintains a global map for the robots and each of the robots maintains a submap of the features in his immediate vicinity. Each of such submaps has its own reference frame where its relative position to the global map is known. At intervals, the robots update the global map with the features present in his submap. A new reference frame is created using the robot's last position as the estimate of the relative pose of the local frame within the global frame of reference. This method also establishes a new reference frame for the robot whose initial starting position is not known by building a correspondence set between the robot's local map and the global map and using a pair of features to determine an estimate of the rotation and translation.

In [50], Howard extends the single-robot Rao-Blackwellized particle filter to handle the multi-robot SLAM problem, where robots' relative poses are not known a priori. To solve the problem, this method assumes the robots can detect, identify each other, and measure the relative pose of other robots when they bump into one another. Once the relative poses are determined, the particle filter is initialized using the determined poses. Then the common map is updated using the observations from both robots. The method also revises the filter to support time-reversed updates. When robots encounter each other, they update the map with then-current sensor information. Then, in time-reversed order, the previous sensor measurements of the robots are fed to the filter

to update the map. This method uses laser-stabilized odometry, in which laser range data is used to correct the raw odometry estimate. The approach presented in this thesis, builds upon this work.

Andrade-Cetto [46] solves the multi-robot SLAM problem using a fully observable EKF and a controller that generates necessary control commands to follow a higher level planned trajectory accurately. The controller uses a nonlinear control technique called Feedback Linearization for multi-robot trajectory tracking. The feedback linearization controls nonlinear systems by algebraically transforming the system dynamics into a linear one [46]. Here linearization is achieved by linear approximations of the dynamics. Trajectories are generated online or to maximize exploration gain. Control errors are not coupled to estimation error, and stability for both the controller and the estimator is achieved. This stability guarantees the system will close the perception-action loop.

Some algorithms [43, 48] use rendezvous strategies and let robots meet at a location to determine their relative locations. In [48], individual robots map and explore areas separately until their meeting. When they meet, one robot receives sensor data from the other robot and estimates their relative location in its own map using Markov localization. In the case of uncertainty, the robot builds a hypothesis on their relative location and tries to meet at the expected location. If they don't meet, the hypothesis is rejected and the robots continue with hypothesis generation about their relative location. Once they meet, the robots matches the partial maps on the basis of geometric information they contain, especially doors, junctions and corners, then share their maps

and perform coordinated exploration. The methods introduced in [43, 44] are similar to [48] except the algorithms use a particle filter to estimate the position of one robot in the other robot's partial map.

The particle filter in [3] can localize a robot inside or outside a map, while in [44] the particle filter only considers robot locations that are part of trajectories that overlap with the partial map. Robots in [47] perform incremental localization and mapping independently. Then when two robots sight each other, the algorithm uses the robots as unambiguous landmarks for map merging and a correspondence is established between two points on the manifold. Then, the robots arrange a rendezvous at the two points on the manifold to verify a correspondence.

2.6 Exploration

An efficient coordination is necessary when a team of robots are deployed to map an unknown environment. If the robots know their relative positions and have a shared map, then effective coordination can be achieved by guiding the robots into non-overlapping areas of the environment. This is done by assigning the robots to different exploration frontiers, which are borders of the partial map at which explored free-space is next to unexplored areas [44, 45, 51, and 52]. A behavior based approach is used for multi-robot exploration in [49]. Robots know their initial relative poses and start exploration while updating the shared grid-base map. It uses potential fields to coordinate exploration among the robots. The robots have 360-degree sensor coverage and perform trilateration through the use of the distance sensor. The schema parameters are used to control such schema to maintain accuracy in SLAM. The function is composed of

attractive force towards unexplored frontiers and repulsive force against obstacles and against other robots. Thrun et.al [23] focuses on reducing the odometry error during exploration [51]. The algorithm separates the environment into stripes. When one robot moves, the other robots remain stationary, tracking the moving robot. The robots are forced to stay close to each other within visibility range. More sophisticated exploration techniques have been introduced in [52]. In [52], robots share a common map and estimate the cost to reach frontier cells and the utility of them. Then each robot places a bid based on the estimates, and then the central mapper assigns the task to individual robots based on the bids. Burgard et. al [51] also take into account of the costs of reaching a target point and the utility of target points for the multi-robot coordination, assuming the robots know their relative positions.

3. Methodology

The previous chapter presented background information on robot localization and mapping and the SLAM problem. This chapter builds on this foundation in discussing the mathematical formulation of an algorithm to perform multi-robot FastSLAM for large domains.

3.1 Mathematical Formulation for Multi-robot Fast SLAM

The original FastSLAM algorithm was developed for single-robot mapping and maintains a particle filter to represent the posterior distribution of a robot along with a map distribution. Each particle of the filter represents a possible path of the robot and maintains a map according to the path estimation. Updating the filter given a new motion control and observation is done in four steps as shown in Figure 3.1.

```
1. Sample a new robot pose given the new control
2. Update maps corresponding to the new observation
3. Assign a new weight to the particles
4. Resample the particles based on their weights
```

Figure 3.1: Basic FastSLAM Algorithm

The first step is to propose a new robot pose for each particle that is consistent with the previous pose estimate and the new robot motion command. Next, the map of each particle is updated incorporating the new sensor observation. Then an importance weight is assigned to each particle based on consistency of the map with the new sensor incorporations. Lastly, a set of samples are drawn from the old set of samples based on

the newly calculated weight. At the end of mapping, the map of the particle with the highest weight is selected to be presented.

This SLAM method is extended to multi-robot SLAM allowing robots to have the capability to detect each other, and exchange their data during their exploration. In this case, a robot has to simply maintain a number of filters matching the number of robots it has come in contact with during exploration: one for his data and one for the data from each one of the robots.

I will begin with formulating mathematical theory for multi-robot FastSLAM with known initial robot poses in Section 3.1.1, and then for multi-robot FastSLAM with unknown initial poses in Section 3.1.2.

3.1.1 Multi-robot fastSLAM with Known Initial Poses

The single-robot FastSLAM notation can easily be extended to multi-robot FastSLAM notation when the initial poses of the robots are known a priori. The robots can set up a number of filters that match the number of robots on the team, and initialize each filter with the known positions. Then as they begin to explore, they can pass their odometry and sensor data to each other and perform a FastSLAM processing of all the data simultaneously. In this case, the robots need a capability to detect each other and have a communication link available during the exploration. If we consider just a two-robot mapping problem where both robots knows their initial relative pose against each other and take steps in lock-step time, then the two-robot SLAM for a robot, identified as robot1, becomes the problem of estimating $p(s_{1:t}^{1}, s_{1:t}^{2}, m | z_{1:i}^{1}, u_{0:t-1}^{1}, s_0^{1}, z_{1:t}^{2}, u_{0:t-1}^{2}, s_0^{2})$ which can be expressed as a product of three factors as

$$p(s_{1:t}^{1}, s_{1:t}^{2}, m | z_{1:t}^{1}, u_{0:t-1}^{1}, s_0^{1}, z_{1:t}^{2}, u_{0:t-1}^{2}, s_0^{2}) =$$

$$p(m | s_{1:t}^{1}, z_{1:t}^{1}, s_{1:t}^{2}, z_{1:t}^{2}) \cdot p(s_{1:t}^{1} | z_{1:t}^{1}, u_{0:t-1}^{1}, s_0^{1}) \cdot p(s_{1:t}^{2} | z_{1:t}^{2}, u_{0:t-1}^{2}, s_0^{2}) \quad (3.1)$$

where the superscript numbers identify the robots.

The first term describes the distribution over maps, and the second and the third terms describe the posterior distributions for the robots. The superscript number denotes the identity of the robots. Thus $s^{1}_{1:t}$ denotes a trajectory of robot 1 from startup to time t and the remaining letters convey the same meaning as with the single-robot SLAM formalism, with the identity of a robot added. Notice this factorization assumes that these trajectories are independent from each other and the observation made by one robot is not dependent on the other robot's pose. Although this assumption doesn't hold true in some cases, it is generally true when robots are far apart. When a robot's pose influences the observation of the other robot, the observation influenced can be discarded after determining the relative pose of the influencing robot. For simplification, this assumption was made in formalizing the multi-robot SLAM solution. Figure 3.2 shows the Bayes net for multi-robot SLAM with known initial poses [50].

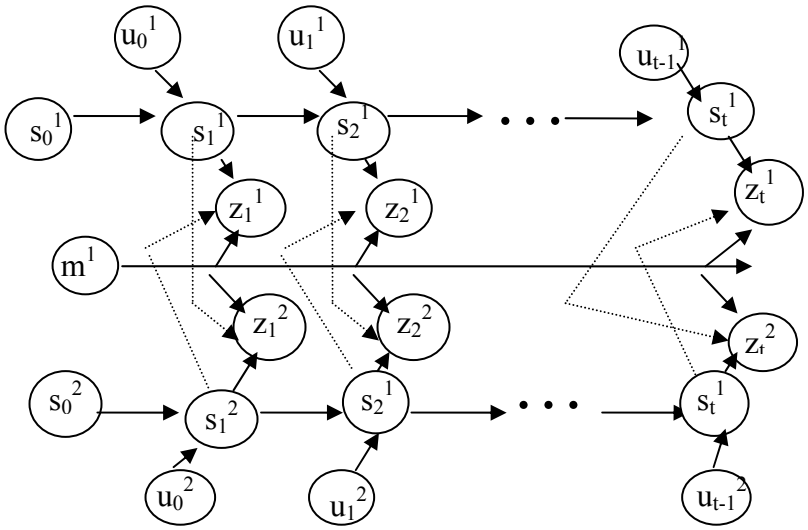


Figure 3.2: Bayes net of robot 1 for multi-robot SLAM with known initial poses

Dotted lines show ignored dependencies between the observations made by one robot and the pose of the other robot. The superscript identifies the robots in which 1 indicates the master robot (the robot whose point of view is focused on in this discussion) and 2 the virtual robot which represents the encountered robot.

The filter's update functions for this case are given as follows:

$$s_t^{1(i)} = A(u_{t-1}^1, s_{t-1}^{1(i)}, z_t^1) \tag{3.2}$$

$$m_t^{1(i)} = M(s_t^{1(i)}, z_t^1) + M(s_t^{2(i)}, z_t^2) + m_{t-1}^{1(i)} \tag{3.3}$$

$$w_t^{1(i)} = S(z_t^1, s_t^{1(i)}, m_{t-1}^{1(i)}) S(z_t^2, s_t^{2(i)}, m_{t-1}^{1(i)}) w_{t-1}^{1(i)} \tag{3.4}$$

where A, M, and S are the action, map, and sensor models, respectively. This method has two limitations for practical implementation. First, the state space induced by two robot position estimation is larger than that of the single robot case, and this causes sparse sampling. To compensate for this problem, it is necessary to increase the number of particles which requires more memory space allocated for the mapping program.

However, this problem doesn't exist when the master robot doesn't explore the area that is already explored by the other robot, because the total path length would scale down to half the length that would have taken for the single robot case. Second, during the filter resampling process, particle impoverishment occurs in the vicinity of a stationary robot if the robot stops moving. Particle impoverishment causes the filter to diverge. Therefore, best results are obtained when both robots are moving at comparable speeds.

3.1.2 Multi-robot FastSLAM with Unknown Initial Poses

In the mathematical formulation for multi-robot FastSLAM with unknown initial poses, it is assumed that robots can detect each other when they meet, measure their relative poses, and pass their range and odometry data from their last encounter up to their latest meeting time.

Initially each robot is placed in unknown locations and begins building maps in their local coordinate systems. When two robots encounter one another, they both measure the other robot's relative location and then exchange their motor command history along with their observation data. Once data is obtained, each robot sets up a virtual robot with a separate local grid map, initializes it to the relative location within the separate local map, and lets the virtual robot perform FastSLAM on the exchanged data while the master robot continues its exploration. The exchanged data is passed to the virtual robot in the time-reversed order. Then both the robots update their own map by incorporating data from the other robot as if a virtual robot is following the steps taken by the robot backward.

Figure 3.3 shows the Bayes' net for multi-robot SLAM with unknown initial pose where Δ denotes the relative pose of the robots. The superscript identifies the robots in which 1 means a master robot and 2 means a virtual robot.

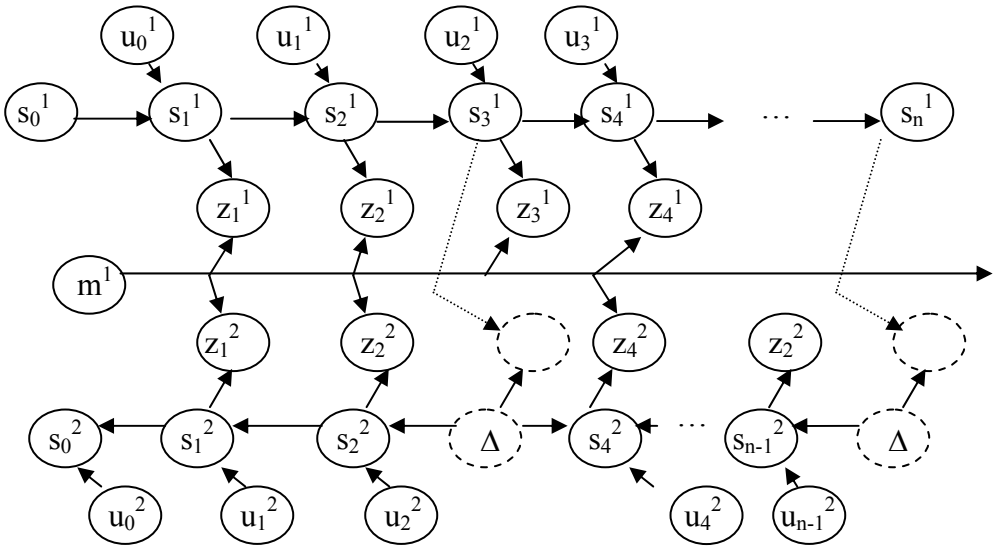


Figure 3.3: Bayes net for multi-robot SLAM with unknown initial poses

After the exchange of data, both robots remember the location and time of their meeting and the identity of the encountered robot and continue exploration. When they come across again, they both exchange the data from their last meeting location up till the current time. This process repeats at each encounter.

Considering just two-robot SLAM, the problem for robot 1 is to estimate $p(s_{1:t}^1, s_{k-1:1}^2, m | z_{1:t}^1, u_{0:t-1}^1, s_0^1, z_{k-1:1}^2, u_{k-1:1}^2, \Delta_k^2)$ after the first encounter with robot 2. Here, Δ_k denotes

for the relative pose of the robots at the encounter time k. This expression can be rewritten as

$$p(s_{1:t}^{1}, s_{k-1:1}^{2}, m^{1} | z_{1:t}^{1}, u_{0:t-1}^{1}, s_{0}^{1}, z_{k-1:1}^{2}, u_{k-1:1}^{2}, \Delta_k) =$$

$$p(m^{1} | s_{1:t}^{1}, z_{1:t}^{1}, s_{k-1:1}^{2}, z_{k-1:1}^{2}) \cdot p(s_{1:t}^{1} | z_{1:t}^{1}, u_{0:t-1}^{1}, s_{0}^{1}) \cdot p(s_{k-1:1}^{2} | z_{k-1:1}^{2}, u_{k-1:1}^{2}, s_{k}^{1}, \Delta_k) \quad (3.5)$$

and the problem for robot 2 is to estimate $p(s_{1:t}^{2}, s_{k-1:1}^{1}, m | z_{1:t}^{2}, u_{0:t-1}^{2}, s_{0}^{2}, z_{k-1:1}^{1}, u_{k-1:1}^{1}, \Delta_k)$ after the first encounter with robot 1. This expression can be rewritten as:

$$p(s_{1:t}^{2}, s_{k-1:1}^{1}, m^{2} | z_{1:t}^{2}, u_{0:t-1}^{2}, s_{0}^{2}, z_{k-1:1}^{1}, u_{k-1:1}^{1}, \Delta_k) =$$

$$p(m_2 | s_{1:t}^{2}, z_{1:t}^{2}, s_{k-1:1}^{1}, z_{k-1:1}^{1}) \cdot p(s_{1:t}^{2} | z_{1:t}^{2}, u_{0:t-1}^{2}, s_{0}^{2}) \cdot p(s_{k-1:1}^{1} | z_{k-1:1}^{1}, u_{k-1:1}^{1}, s_{k}^{2}, \Delta_k) \quad (3.6)$$

After the robots encounter each other, robot 1 initializes the pose of robot 2 using:

$$s_{k}^{1} = \Delta_k \oplus s_{k}^{2} \quad (3.7)$$

and robot 2 initializes the pose of robot 1 using

$$s_{k}^{2} = \Delta_k \oplus s_{k}^{1} \quad (3.8)$$

where the $\oplus$ operator indicates an coordinate transformation.

The filter of a robot now maintains two instances: one for its own motion and the other for time-reversed motion of the other robot. The filter update function for each particle I of robot 1 therefore can be defined as

$$s_{t}^{1(i)} = A(u_{t-1}^{1}, s_{t-1}^{1(i)}, z_{t}^{1}) \quad (3.9)$$

$$s_{k-1}^{2(i)} = A(u_{k}^{2}, s_{k}^{2(i)}, z_{k-1}^{2}) \quad (3.10)$$

$$m_{t}^{1(i)} = M(s_{t}^{1(i)}, z_{t}^{1}) + M(s_{k-1}^{2(i)}, z_{k-1}^{2}) + m_{t-1}^{1(i)} \quad (3.11)$$

$$w_{t}^{1(i)} = S(z_{t}^{1}, s_{t}^{1(i)}, m_{t-1}^{1(i)}) S(z_{k-1}^{2}, s_{k-1}^{2(i)}, m_{t-1}^{1(i)}) w_{t-1}^{1(i)} \quad (3.12)$$

and for robot 2 as

$$s_{t}^{2(i)} = A(u_{t-1}^{2}, s_{t-1}^{2(i)}, z_{t}^{2}) \quad (3.13)$$

$$s_{k-1}^{1(i)} = \mathrm{A}(u_k^1, s_k^{1(i)}, z_{k-1}^1) \tag{3.14}$$

$$m_t^{2(i)} = \mathrm{M}(s_t^{2(i)}, z_t^2) + \mathrm{M}(s_{k-1}^{1(i)}, z_{k-1}^1) + m_{t-1}^{2(i)} \tag{3.15}$$

$$w_t^{2(i)} = \mathrm{S}(z_t^2, s_t^{2(i)}, m_{t-1}^{2(i)})\mathrm{S}(z_{k-1}^1, s_{k-1}^{1(i)}, m_{t-1}^{2(i)}) w_{t-1}^{2(i)} \tag{3.16}$$

where A stands for a new-reversed action model due to the robot moves backwards and M, and S are the map, and sensor models, respectively and 1 and 2 identify actual robots. The filter set up here updates the causal and acausal instances at the same rate.

3.2 Multi-robot FastSLAM for Large Domains

FastSLAM algorithm lets a large number of particles take individual action, each one of them drawing a map of the environment. Each particle maintains a global map during its life cycle. When a robot completes exploration of the environment, it selects the map of the best particle to keep. However, letting each particle maintain a global map may constrain the performance of the FastSLAM algorithm in mapping a large area. To overcome this constraint, letting each particle maintain only a local area at one time is an option.

In this FastSLAM implementation, each robot begins mapping on its own local grid map. When robots meet, robots first establishes their relative locations, pass their collection of data to one another, and then each one of them sets up a virtual robot with another local grid map, initialize the virtual robot within the second map to the measured relative pose. The master robot and the virtual robot perform FastSLAM at the same rate, where the master robot performs FastSLAM with newly collected data and the virtual robot with the passed data in the reversed order. Periodically each particle of the filters for the master and virtual robots transforms the grid map into an ASR data, attaches the

posterior of the robot it represents and saves it for loop closing. Then each particle initializes its local grid map with the previous grid map in the overlapping area, although it would be not be necessary to begin mapping without any data import from the previous map. However, I have not tested the latter method to verify the idea.

The master and virtual robots generate their own ASRs and, in the end of mapping, these ASRs are combined to generate a global map. However, while mapping, the master robot and virtual robot consult their own ASRs and the ASRs of the other to improve accuracy of mapping and to perform loop closing. Figure 3.4 summarizes the multi-robot FastSLAM algorithm for large domains.

The implementation of FastSLAM for large domains built on the existing FastSLAM implementation found in "pmap" at [69]. The "pmap" algorithm uses regular, global grids for each particle. This algorithm is altered to use local grids and global ASRs instead of global grid maps for the particles in order to conserve memory. Therefore, many of basic functions used in the implementation are from the "pmap" program, especially the resampling method and error calculation method required to assign weights to the particles. The remaining sections of Chapter 3 briefly discuss the Absolute Space Representation and the basic functions taken from the "pmap" implementation [69] along with other methods developed for this thesis.

```
Multi-Robot FastSLAM:

  1. Set up a filter (numOfParticles)
  2. For each particle (initial_robotpose, odom_pose, a
     lidar scan)
     2.1 Set up a grid map
     2.2 Initialize sample poses (initial_robotpose,
           odom_pose, a lidar scan)
     2.3. Add ranges to the maps at the initial robot
     pose
  3. Resample the samples
  4. Update (new_odom_pose, a lidar scan)
     For each particle:
     4.1 Propose a new robot pose
     4.2 Update local map with lidar_scan
     4.3 Update the weight
  5. Resample the samples
  6. If travel distance > 10m, go to step 7. Else go to
     step 9
  7. Transform grid into an ASR
  8. Attach global window to the ASR
  9. Attach id to the ASR and save it.
  10. If master robot and met another robot, go to step
      12. Else go to step 11
  11. If not done, go to step 4. Otherwise, go to step
      15
  12. Determine the relative pose
  13. Set up a virtual robot
  14. Go to step 1
  15. Combine ASRs into a global map
```

Figure 3.4: Multi-robot FastSLAM Algorithm

3.3 Absolute Space Representation

Absolute Space Representation expresses the world as a graph. Absolute space is termed from the idea that each space is independent of all other spaces. In Absolute

Space Representation, regions are represented as nodes and access points as edges. Figure 3.5 shows an ASR example.

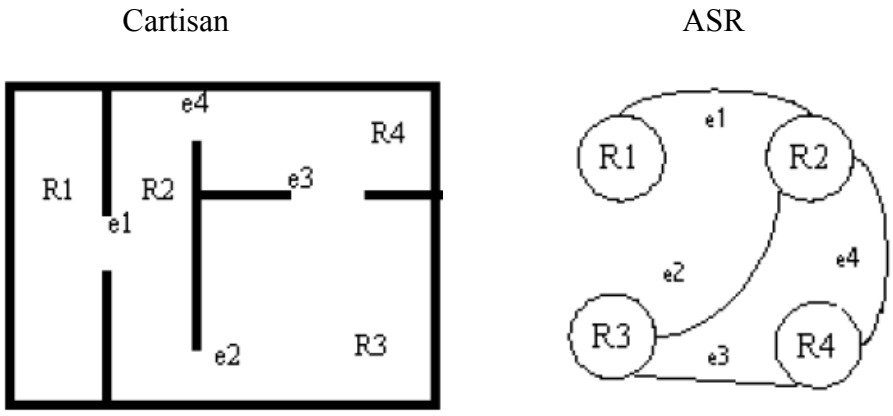


Figure 3.5: An ASR Example

Here nodes R1, R2, R3, and R4 represent individual areas and connected through shared access points that are represented by edges in the graph. Absolute Space Representation becomes beneficial especially when the structure of the environment to be mapped is composed of mostly straight lines without many obstacles within. In my implementation, as a robot continues mapping, inactive ASR maps are stored away. When the robot comes into the area that a stored ASR map occupies, the robot simply opens the ASR map and continues mapping. An ASR is made of polylines, and a polyline is composed of a series of connected vertices. However, when a portion of the polyline is straight, the straight line can be represented by just two end points. This simplification method is beneficial when an environment is composed mostly of straight lines and the range sensor's distance readings are fairly accurate. This is the case in most buildings and

man made environments. The AFIT bldg 640 is rectangular shape and the SICK scanning lidar sensor on the Pioneer P2-AT8 is known for detecting distances to the structures very closely. This enables one to extract polylines from a grid without any special mathematical calculations. Absolute space representation of an area benefits when the number of unvisited cells outnumber the number of visited cells on the order of magnitude.

Although an ASR can exist absolutely irrelevant to other ASR maps, each ASR map from the implementation carries an offset value to indicate the location of the ASR map within the global map. Thus, adding the offset to the vertices of an ASR positions the ASR in the global map. The structure of an ASR map representation is displayed in Appendix A.4

The next section discusses each part of the implementation in detail including the action model, resampling method, and map transformations.

3.4 Implementation Algorithms

This section discusses all the algorithms that are used for my implementation.

3.4.1 Lidar-Corrected Robot Poses

The robot's raw odometry readings contain a great deal of errors. Figure 3.7 shows the blueprint of second mezzanine floor of bldg 640, AFIT, and Figure 3.6a shows a map drawn using only pure odometry readings. The map is far from reality. Therefore, using the raw odometry reading for performing FastSLAM requires a large number of particles to be used to explore a wider range of errors. Since lidar's distance readings are

very accurate, the algorithm takes advantage of it by prescreening robot odometry poses using a series of lined lidar scans. Then these lidar-corrected robot odometry readings are passed to the algorithm and used by FastSLAM. The lidar-correction algorithm that I used is provided by the "pmap" program [69]. Figure 3.6b shows a map of the same area drawn by using lidar-corrected odometry poses.

Figure 3.6a: Naïve map

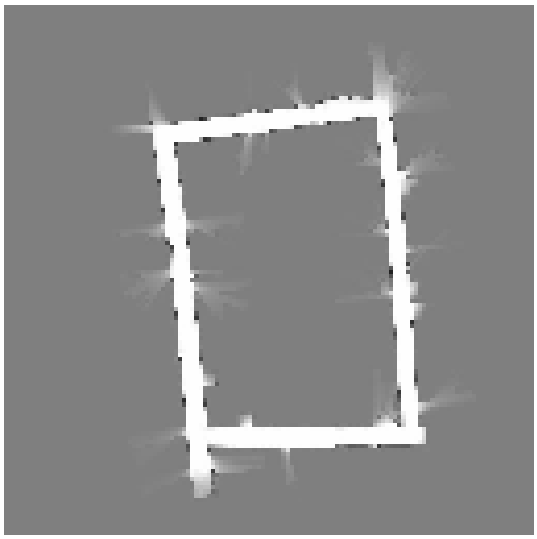


Figure 3.6b: Map with lidar-correction

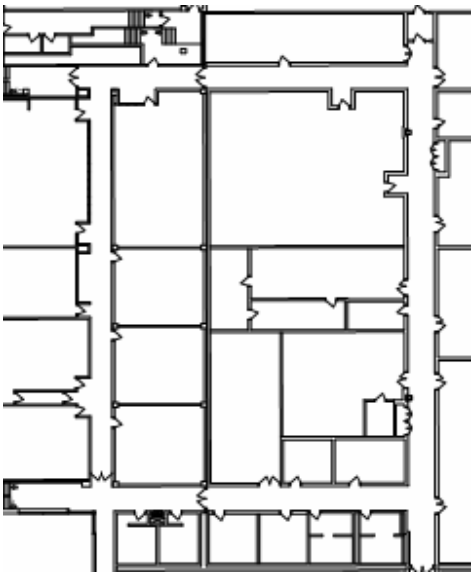


Figure 3.7: Blueprint of the second floor of bldg 640, AFIT

As can be seen in the comparison made in Figure 3.6a and Figure 3.6b, the map with lidar-corrected robot poses is more accurate than map with the raw odometry readings, it is still subject to errors, thus requiring a SLAM technique. Section 3.4.2 discusses the action model required for a FastSLAM algorithm.

3.4.2 Action Model for the Robots

The FastSLAM algorithm is probabilistic and requires an action model for the particles. Each particle moves according to this action model while accounting for uncertainty in robots movement by including a margin in their action. In the implementation, the action model is a 3 x 3 matrix. If the action model is denoted as *A*, *A* is multiplied to the pose difference δ between two consecutive steps to obtain a 3-dimetional vector σ. Each element of σ is then fed into a random Gaussian number generator as a standard deviation to obtain a random value to account for uncertainty in

each dimension of a robot pose for a sample *i*. The action model used for the robots in my implementation is shown in Table 3.1.

Table 3.1: Action model

	Uncertainty caused by movement in x direction	Uncertainty caused by movement in y direction	Uncertainty caused by rotation
Uncertainty in x direction	0.075	0.015	0.09
Uncertainty in y direction	0.015	0.075	0.09
Uncertainty in rotation	0.035	0.035	0.07

The first and the second cell of the first row represent the uncertainty in a robot's x-coordinate pose estimation caused by the robot's pose change in x-coordinate and y-coordinate directions, respectively. The last cell of the same row accounts for the uncertainty in a robot's x-coordinate pose estimation cause by the robot's orientation change. First and second rows of the action model accounts for the uncertainty in x-, y-coordinates caused by a robot's pose change, while the last row accounts for the uncertainty in the orientation caused by the robots pose change.

In the implementation, a robot first determines the difference, $\delta=[dx,dy,d\theta]^T$, between its current and previous lidar-corrected poses. A robot pose is a three-dimensional vector, $[x,y,\theta]^T$, where x and y denotes for x and y coordinates and θ robot's orientation angle. A turn to the robot's right side decreases θ and a turn to its left side increases θ. The action model is multiplied to δ vector to obtain a σ vector. After that, each sample obtains three random numbers obtained from a normal distribution with sigma σ for each element of a pose vector. These elements are added to the delta and the

previous pose and the robots next pose for the sample is set to this new location. Summarizing this notation in formulas, we get

$$\delta.x = 0.075*dx + 0.015*dy + 0.09*d\theta \quad (3.17)$$

$$\delta.y = 0.015*dx + 0.075*dy + 0.09*d\theta \quad (3.18)$$

$$\delta.r = 0.035*dx + 0.035*dy + 0.07*d\theta \quad (3.19)$$

$$x' = x + dx + N(0, \delta.x) \quad (3.20)$$

$$y' = y + dy + N(0, \delta.y) \quad (3.21)$$

$$\theta = \theta + d\theta + N(0, \delta.\theta) \quad (3.22)$$

Here, x',y', and θ' denotes for three elements of a new robot pose vector, and the operators + and * implies an vector addition and a multiplication, respectively. The variables $\delta.x$, $\delta.y$, and $\delta.y$ indicate elements of the δ vector in order. When two poses vector a and b are given, a new pose vector c is obtained through the following operations:

$$c.x = \cos(b.\theta)*a.x - \sin(b.\theta)*a.y + b.x \quad (3.23)$$

$$c.y = \sin(b.\theta)*a.x + \cos(b.\theta)*a.y + b.y \quad (3.24)$$

$$c.r = \text{atan2}(\sin(a.\theta+b.v), \cos(a.\theta+b.\theta)) \quad (3.25)$$

Atan2 is a Linux function that returns the arc tangent of the two variables. Figure 3.8 summarizes how a new robot pose is estimated by a particle.

```
Robot Pose Estimation Method:
   1. Initial sample pose= the intial robot pose if the
      pose is known, otherwise to (0,0,0)
   2. Odom_pose=Lidar-corrected pose=1st odemetry reading
   3. Calculate Delta=old odom_pose – new_odom_pose
   4.Per Particle i:
      4.1. Sigma(i)=Action_model * delta
      4.2. Sample pose(i)=delta + gaussian random(sigma)
```

Figure 3.8: A new robot pose estimation algorithm

Once the robot's pose is determined, each sample updates its map according to the newly proposed position. The next section discusses how a map is updated with a lidar scan.

3.4.3 Map Update Method

Updating a map is done by simply tracing the rays of the ranges once the new pose of the robot for a sample is determined. Once a robot's newly estimated pose is ready, the global location of the range is determined by extending it from the current robot's estimated pose within a global, virtual grid map. Once the horizontal and vertical cell indices were determined for the global grid, the cell indices are translated into local grid cell indices by accounting for the global location of the local grid map. Figure 3.9 summarizes the map update function. It also shows how the weight of a sample is calculated.

```
Map Update Method Algorithm:
  For each sample
     1. poses[step]=current_sample_pose
     2. Resample Samples
     3. err=0;
     4. For each range
        4.1. Get the global cell position of the range
        4.2 Transform the global cell position to the
              local cell position
        4.3. Increase the value of the cell by 1
             If value>127, value=127
        4.4. Find min_d=the minimum distance to the
              nearest occupied cell
        4.5. Calculate err+=min_d*grid_resolution
     5. w(i)=err*err
```

Figure 3.9: Map update method algorithm

After the sample updates its map with range information, the sample determines the distance to the nearest occupied cell from the newly updated cell. This global distance is summed over for a range scan.

Then after incorporating all the ranges of a scan, this summed distance is squared and added to the weight of the sample. The weight of a sample is reset to zero after a resampling.

After all the samples update their maps and new importance factors are calculated, only samples with good map survive to the next generation through resampling. The following section discusses the resampling technique used in the implementation.

3.4.4 Resampling Technique

Each sample carries a weight based on its accuracy. The weights represent the error of the map for each of the samples. After each sample determines the robot's next

pose and updates its map based on its randomly estimated pose calculated via the action model, it calculates its error in its mapping. After each sample determines and updates the location of the cell that a range reading indicates, it determines the distance of the nearest occupied cell from the updated cell. This distance is then added to the weight of the sample. Therefore, a higher weight means a less accurate map for the sample. This weight value is turned into a probability through

$$p(i) = \exp\left(-\left[\frac{w(i)}{\sigma^2} \right] \right) \tag{3.26}$$

Where σ^2 represents the resampling variance and $w(i)$ stands for the weight of the i^{th} sample. The sample probability values are normalized and used for resampling. The resampling technique is Walker's algorithm that is provided in the pmap program. Walker's algorithm [70] produces a value k consistent with its probability, given K discrete events with different probabilities $p[k]$. This algorithm performs preprocessing on the probabilities of the samples, and then provides two arrays: floating point $F[k]$ and integer $A[k]$. A value k is chosen from $[0..K-1]$ with equal likelihood, and then a uniform random number u is compared to $F[k]$. If it is less than $F[k]$, then k is returned. Otherwise, $A[k]$ is returned.

When a best local map is selected, the algorithm extracts polylines from the sparse grid. The following section discusses the extraction method.

3.4.5 Transformation of a Grid into an ASR

The algorithm first cleans the grid that is to be converted to an ASR by resetting the cells with values less than a user specified threshold. Then starting from (0,0) of the

local grid, the first cell with a value higher than the threshold when moving in x-direction is selected as the beginning vertetex a new polyline. From this cell, all the neighboring cells whose x, y coordinate values differ by 1 from the starting cell are searched to determine the next vertex to be added to the polyline. Then the next search focuses on the neighboring cells of the newly selected vertex. This process is repeated until there are no connecting cells. When a vertex is selected to be inserted into a polyline, the value of the corresponding cell is set to 0. Extracting polylines continue until all the values in the grid are below the threshold. Appendix A.3 shows the algorithm.

Once a polyline has been constructed, it is sent to a simplifying algorithm that removes vertices lying on a straight line, resolving multiple lines into their two end points. The simplifying algorithm compares the angles of two connected vectors built by three consecutive vertices, and if these angles are close to each other, the middle vertex is removed. The angle of a vector built by two vertices (x_1, y_1) and (x_2, y_2) can be calculated as follows:

$$d = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.27}$$

$$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.28}$$

$$\theta = \arccos\left(\frac{d}{r}\right) \tag{3.29}$$

After all the polylines are extracted, they are saved as an ASR to be reopened when a robot revisits the area. As the robot moves, each sample constantly refers to its stored ASR maps for loop closing. Once overlapping areas are found, it converts the stored ASRs to local grids and initializes the active local map of the sample with the

converted grid for overlapping areas. Since each ASR carries its global window information, determining overlapping ASRs is performed by comparing the windows. Reconstructing a local grid from an ASR is done by determining the cells each polyline lies on and setting the values of the cells to the highest value among the two end vertices values. The algorithm is presented in Appendix A.2.

This chapter presented the methods adopted for the implementation of multi-robot FastSLAM using Absolute Space Representations. Specifically, how a particle filter estimates a robot pose, how the weight of a sample is calculated while updating a map, and how samples are resampled were discussed. The next chapter shows testing results of the implementation and the analysis of the testing.

4. Analysis and Results

In this chapter, testing outcomes and an analysis of the testing are provided. In Section 4.1, testing environment and parameters used in the implementation are discussed, and Section 4.2 discusses testing results and provides an analysis of the results. This chapter concludes with a discussion of the overall results.

4.1 Testing Overview

Testing the implementation was done offline using range data from a Pioneer P2-AT8 that was equipped with a SICK scanning lidar sensor that outputs ranges from -90^{o} to $+90^{o}$ in increments of 1^{o}. Three separate data sets were collected by driving the robot moving at approximately 0.2m/sec around the second mezzanine floor and the third floor of bldg 640, at the Air Force Institute of Technology. Two simulations were carried out using the three sets of data on a Pentium 4 with 2.8 GHz CPU. Figure 3.7 shows the blueprint of the second mezzanine floor of the bldg 640 and Figure 4.1 shows the blueprint of the third floor of the same building.

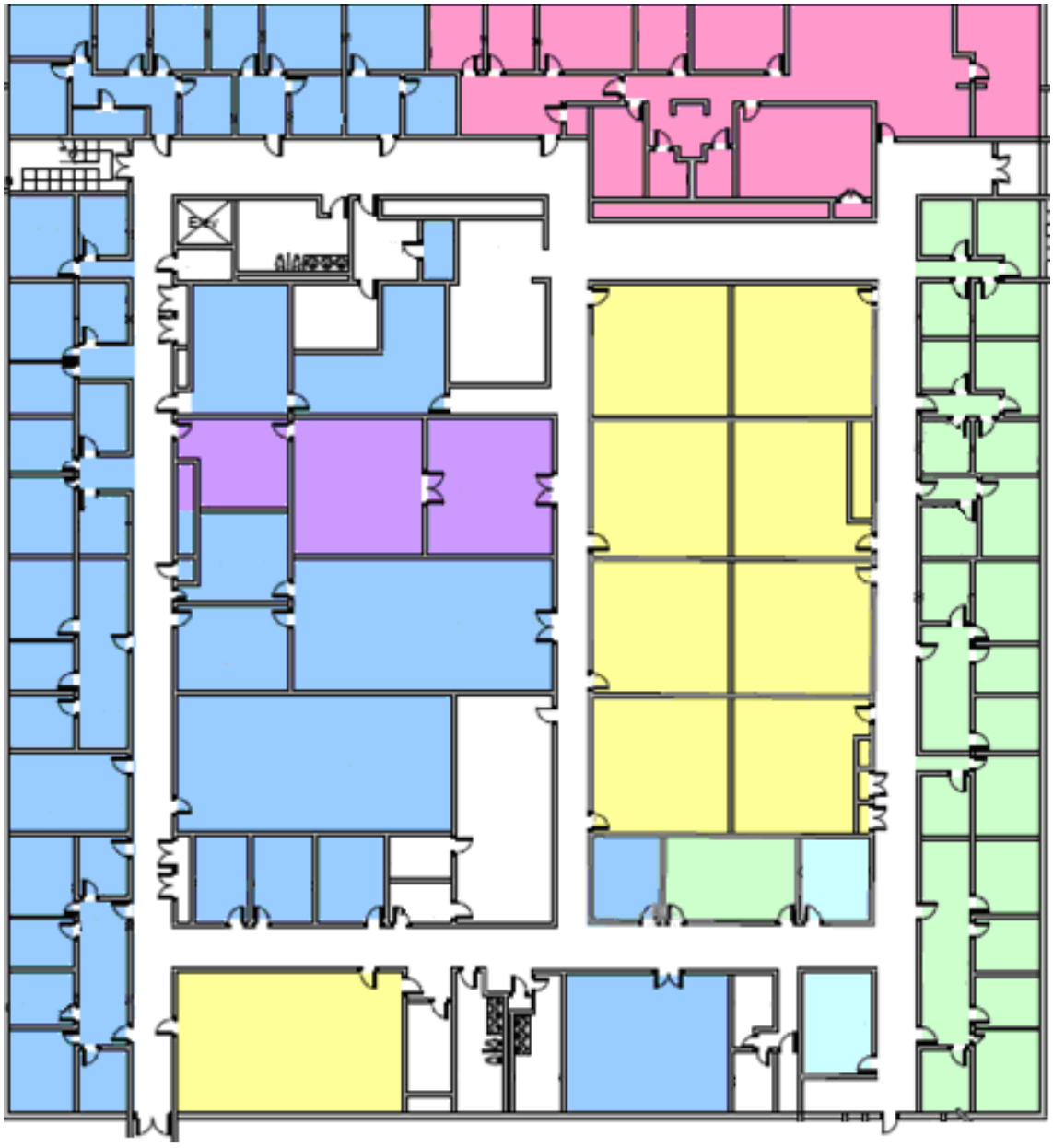


Figure 4.1: Third Floor of Bldg 640.

The dimension of the global map is 64m x 64m while it is 40m x 40m for a local grid map. When a sample begins a local mapping, the robot is always positioned in the middle of the local map. Since I have chosen the cutting travel distance to transform a

grid into an ASR to be 10m, the size of a local map of 40m x 40m seems reasonable to account for the map updates with the ranges read in the end of the round, 10m of travel plus 10m max lidar range in any given direction.

The implementation does not update the maps every time step. Instead, an update is made after the robot has moved 10 cm or turned 15 degrees. These values are chosen arbitrarily. The implementation requires several other parameters to be set to appropriate values before launching a simulation. Table 4.1 shows all of the parameters used in the testing.

Table 4.1: Thresholds and Parameters for the Implementation

T/P	Value	Role
α	0.25m	Determines cell resolution
β	5	Determines occupancy of a cell when a sample calculates its correctness in mapping (integer)
ρ	10	Resample Interval(integer)
ε	0.5m	Max error considered by each sample
κ	0.3m	Threshold for keeping polylines
ν	15	Occupancy value for ASR mapping (integer)
σ	0.2	Resample sigma
γ	10 m	Maximum travel distance in an ASR
λ	400	Number of samples used
χ	0.1m	Maximum distance allowed before an update
θ	15^o	Maximum rotation allowed before an update

Each of these parameters was tested with various values to find the best performing value. While other parameters were set to specific values, only one of these parameters at a time was tested with various values, and the value with best output was chosen. The variable α determines a map resolution, lower α results in a better resolution map. The variable β is a threshold to determine occupancy of a cell when calculating

correctness of a move of a sample after each step move. Lower values perform better with low map resolutions. The number of updates a robot makes before it resamples its particles is denoted as ρ. A smaller value is certainly better to select good samples. However, this causes higher computational time for the program.

The variable ε denotes the maximum error each sample is assumed to make per update. Therefore, at each update, each sample searches areas within ε distance to determine the closest occupied cells. When a polyline is extracted from a grid map, the threshold κ is used to remove the polyline. Any polyline with grid vertex values less than this value is cut off. This serves to simplify the resulting map. When a grid map is ready for transformation into a polygonal map, a cell value less than υ is removed from the local grid map. The variable γ is the maximum distance a robot is allowed to travel within a local map and the variable λ denotes for the number of particles used. The parameters χ and θ stand for the maximum distance and rotation allowed before an update to the map, respectively.

4.2. Testing Results and Analysis

In this section, testing results are presented. Section 4.2.1 describes a simulation with the data set collected from the second floor mezzanine of Bldg 640 at AFIT and Section 4.2.2 presents testing result of a simulation of multi-robot FastSLAM with the data collected from the third floor of the same bldg.

4.2.1 Single-Robot FastSLAM with ASRs

To simulate a single robot FastSLAM, the first data set is fed into the implementation. The robot began collecting data from the left hand corner of the bottom of the map shown in Figure 3.7, first moving in the horizontal direction, and then stopped when it returned to its stating location. When the robot ended its tour, its orientation angle should be off by -90.0 degrees from its original orientation. While traveling around the floor, the robot collected laser-range data at the same time. This data is then used to simulate a single-robot FastSLAM with local grid maps and global ASRs. For the simulation, the robot was positioned at [-10m, -20m, 0^{o}] and was fed with Data 1 read in-order. Figure 4.2 shows a series of ASRs that generated over the course of the simulation.

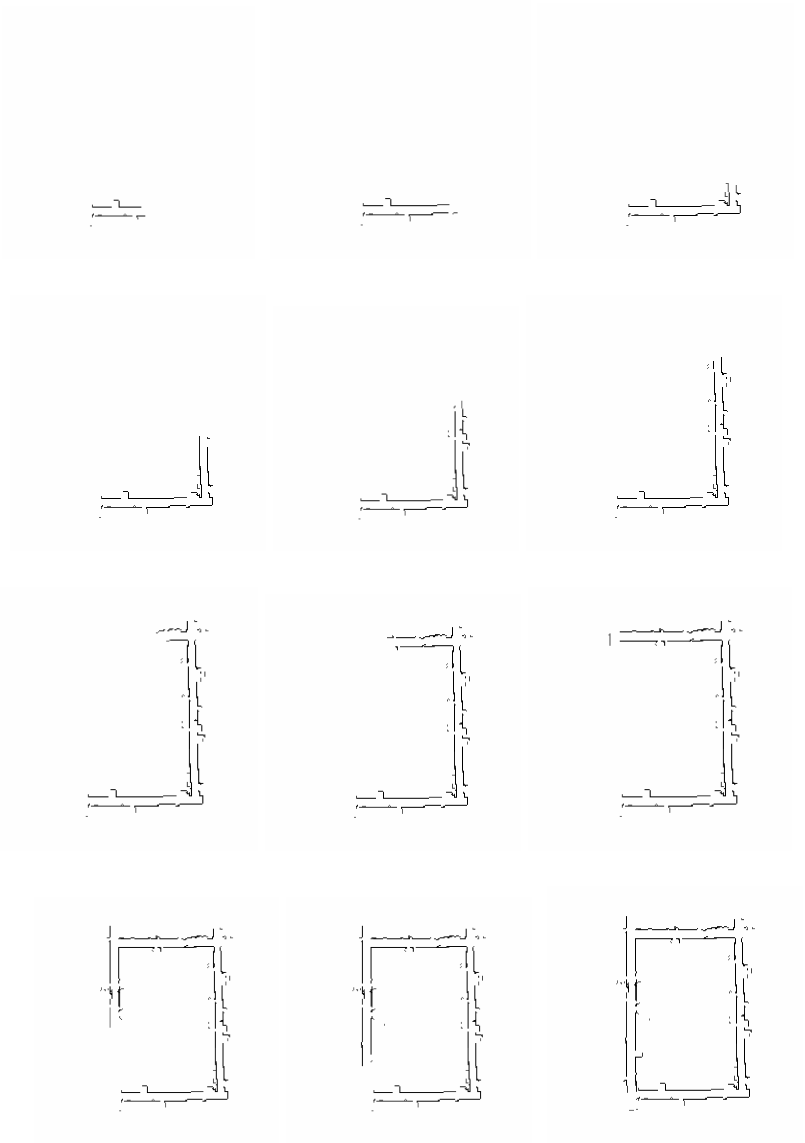


Figure 4.2: ASR maps for data set 1

The ASRs in Figure 4.2 reflect the mapping progress in the simulation.

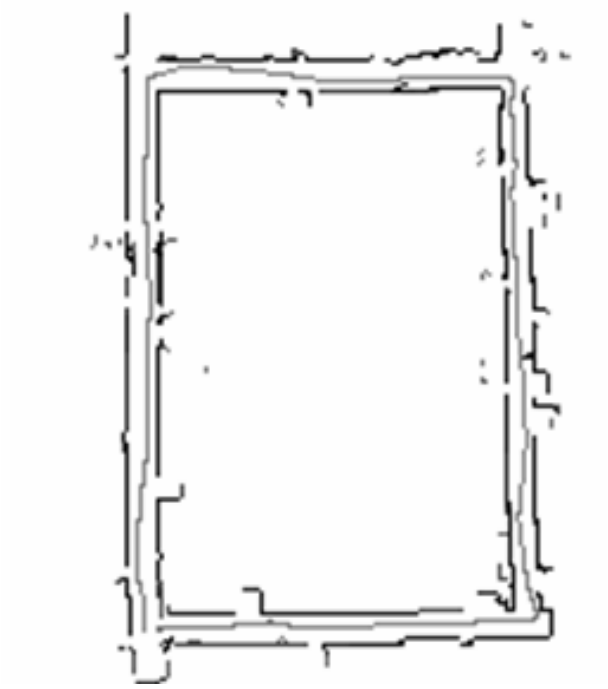


Figure 4.3: Final ASR map with robot poses

Figure 4.3 displays the final ASR with robot poses attached to. The map developed by the implementation for single-robot FastSLAM is very satisfactory. Unlike the maps developed by raw odometry data or by lidar-corrected data only, the ASR map developed by applying FastSLAM shows perfect loop closing and excellent representation of the structures, reflecting the second mezzanine floor of the AFIT engineering building very accurately. Figure 4.4 shows a map in Figure 4.3 overlaid on the blueprint of the building in Figure 3.7.

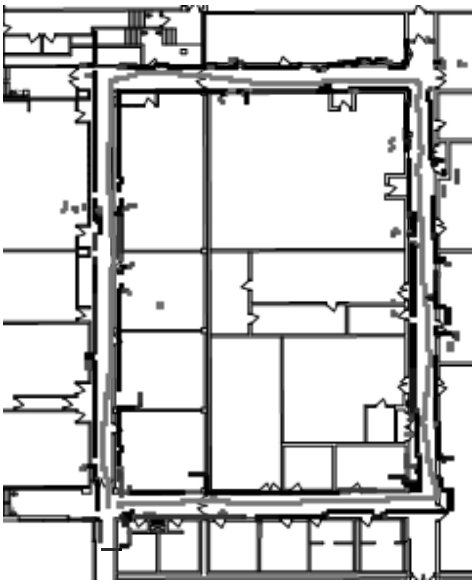


Figure 4.4: ASR Map Overlaid on the Blueprint

The hallways of the map in Figure 4.3 are not distinguishable from the ones of the blueprint. The final map from simulation 1 represents the mapped area perfectly. Figure 4.5 shows a grid map for the same area.

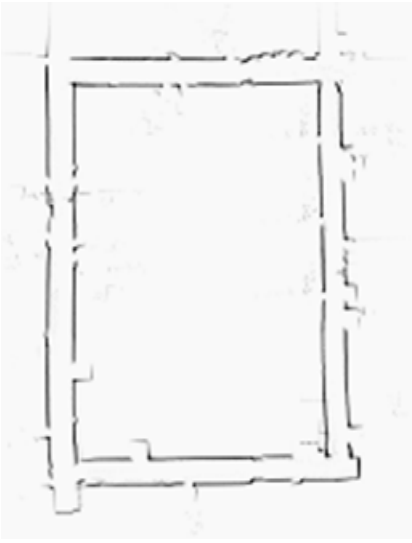


Figure 4.5: Grid Map for Data 1

Although the extracted ASRs in Figure 4.3 outline Figure 4.5 in a brief manner, it shows all the major structures clearly. When more detail is wanted, the threshold value for the extracting algorithm can be lowered, although lowering the threshold would increase the total number of polylines generated. In fact, studying the map, the value of the threshold κ could have been set higher than 0.3m without sacrificing the details of the map. Reducing κ would have reduced the number of polylines within an ASR. Table 4.2 shows the number of polylines and vertices contained in each of ASRs in Figure 4.2.

Table 4.2: Statistics of ASRs in Figure 4.2

ASR ID	Num of Polylines	Num of Vertices
1	9	32
2	6	21
3	13	64
4	18	103
5	27	159
6	28	112
7	34	155
8	34	174
9	38	181
10	16	86
11	30	138
12	29	139
13	20	86
14	21	71
Total	323	1521
Average	23	108.6

The ASRs are identified in time progressing order as displayed in Figure 4.2. The number of polylines and vertices of Table 4.2 also includes overlapped polylines and vertices as well due to the map initialization method used in the implementation when a new local grid is initialized to the previous grid map for overlapping areas. Considering the nature of FastSLAM, this initialization step could have been saved without hurting the quality of the final map to keep the number of polylines and vertices in an ASR to minimum.

The memory space required for the particles of the original FastSLAM algorithm with global grids is 64/.25 x 64/.25 x 400=25 Mbytes to map the second floor mezzanine of Bldg 640 with 400 particles. The memory space used for the particles of the implementation to map the same region is (40/.25 x 40/.25 + 1521 x 16) x 400=19.05 Mbytes. Here, the value of 1521 is the total number of vertices in the global ASR and 16 represents the size of a vertex in bytes in the implementation. The memory space used for the local grids was 9.765 Mbytes and 9.28 Mbytes are used to store a global map representation. The memory space used for the simulation doesn't show much reduction from the one for the grid-based FastSLAM algorithm. However, this is the case because the mapped area was the size of 64m by 64m. Should the mapped area have been larger, the difference in the memory usages would have been much larger due to the local grids having fixed sizes while the grids in the original FastSLAM grow as the mapping area increases. In spite of the small improvement in memory usage, the maximum number of particles that could be used for the simulation was 450 while it was only 303 for the grid-based FastSLAM algorithm.

The simulation for Data 1 is completed in 3.767 minutes. Figure 4.6 displays a map developed by another simulation for the same area. In this simulation, only 200 particles are used while all other parameter setting remained unchanged.

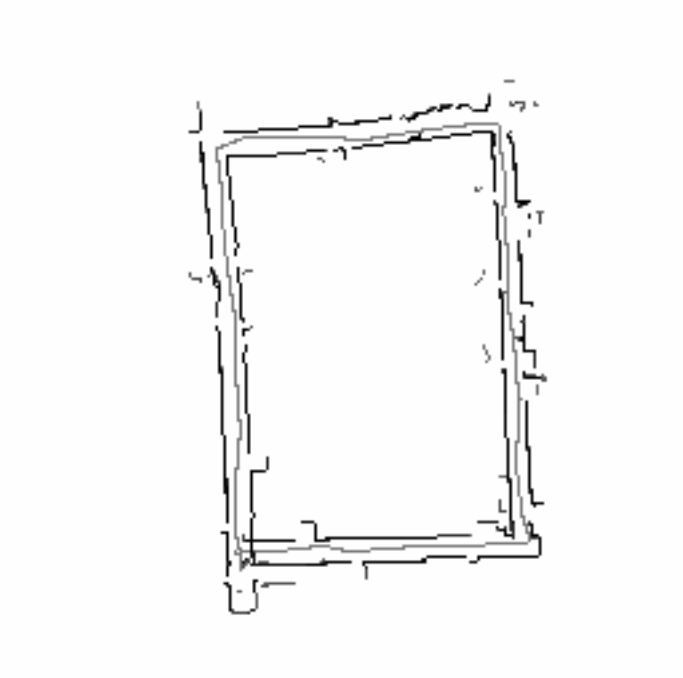


Figure 4.6: ASR Map with 200 Particles

As can be expected, the map is less accurate. The FastSLAM algorithm improves the quality of a map as the robot revisits places. However, improving the quality of a map in this manner can be quite undesirable. Figure 4.7 shows all the polylines of the first ASR from the simulation in time progressing order.

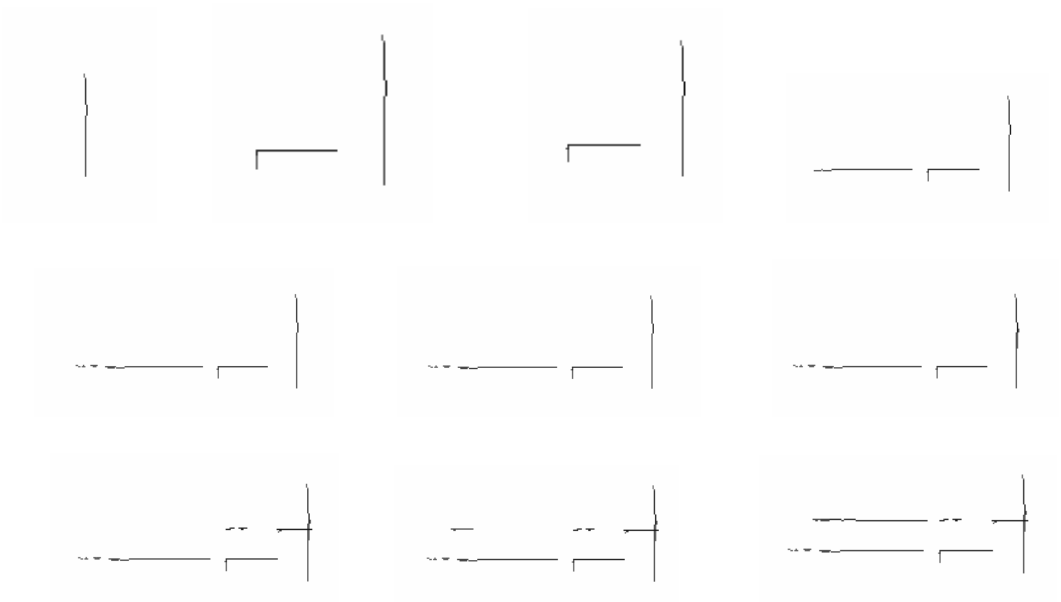


Figure 4.7: Polylines in ASR 1

Some polylines are intelligible due to its size. These small sized polylines can be controlled by adjusting the thresholds. However, the polyline extraction method used for the implementation seems to be very effective, even though very simple in nature. In the next section, a simulation of a multi-robot FastSLAM is done.

4.2.2 Multi-Robot FastSLAM with ASRs for Large-Domain

To simulate 2-robot FastSLAM for large-domain, the third floor of the same building is chosen to collect the data sets, Data2 and Data3, because it has three isles and is wider.

Since only one robot is available and exact pose information of the meeting location of the robots is required, Data 2 and Data 3 are collected in such a manner that the ending location of Data 2 coincides with the starting location of Data 3. The robot is

first placed at the bottom of the middle isle facing upward and then collects Data2 as it goes around the right loop of the floors in Figure 4.1. Once the starting location is reached, the robot is facing toward the left and stopped collecting Data2. The robot then started collecting Data3 by traveling the left half of the area. The robot stopped its travel when it reached the top of the middle isle. These data sets were fed to the FastSLAM algorithm to simulate that Robot1 met Robot2 before Robot1 began exploration and received Data2 from Robot2. In order for the scenario to be realistic, Data2 was fed to Robot2 in the time-reversed order and both Robot1 and Robot2 were positioned at (0m, -20m, 0^{o}).

Figure 4.8 shows the ASRs that generated from the simulation by Robot1 and Figure 4.9 displays the ASRs from Robot2. The ASRs are displayed in a time progressing manner.

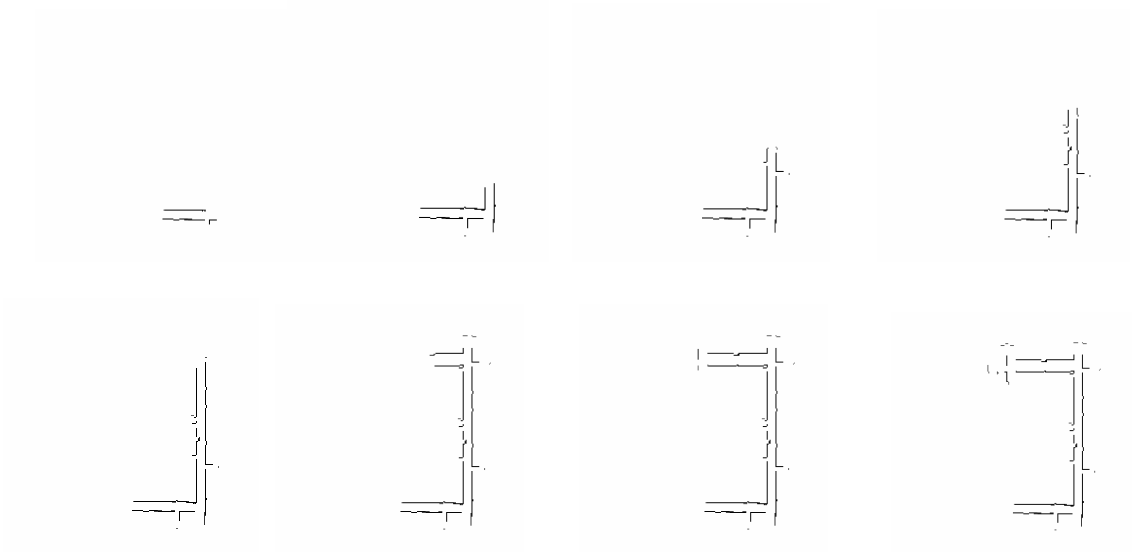


Figure 4.8: ASRs Generated by Robot1

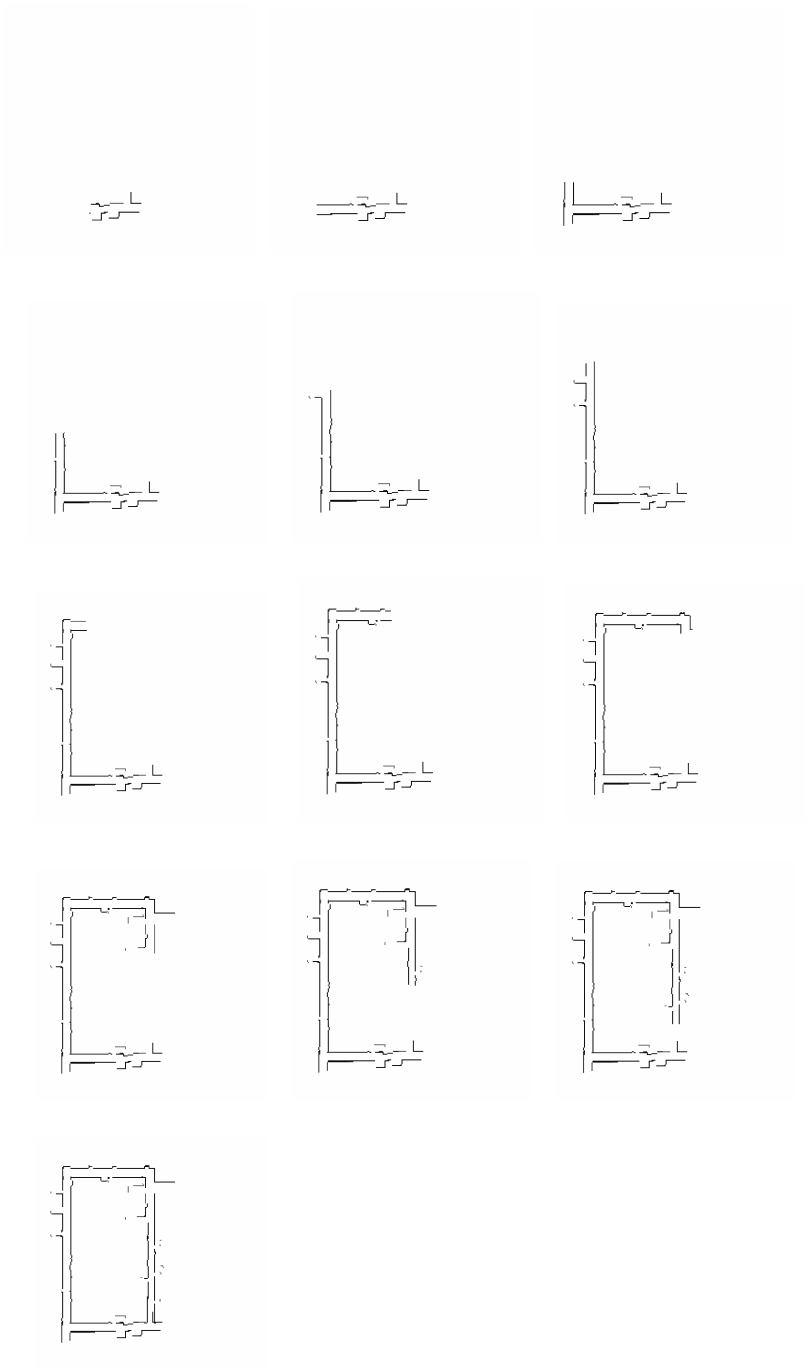


Figure 4.9: ASRs generated by robot2

When the master and virtual robots complete processing all the range data sets, the master robot combines all of the ASRs from itself and the virtual robot and generates a global map. Figure 4.10 presents the global ASR map generated in the end of the simulation. Robot poses for Robot1 and Robot2 are attached to the final output.

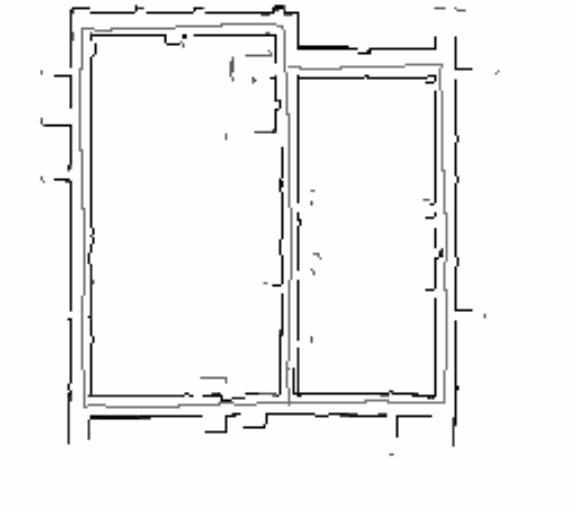


Figure 4.10: Global map generated by ASRs

The resulting map for multi-robot FastSLAM seems to be perfect. The map constructed from the ASRs, shows perfect loop closing and accurate representation of the structures in Figure 4.1. In fact, Figure 4.11 is overlaid on the blueprint in Figure 4.1 to test quality of the map. The ASR map overlays on top of the blueprint perfectly. The hallways of the ASR map are intelligible due to its perfect matching with the ones of the blueprint. Figure 4.3 shows the statistics of second simulation.

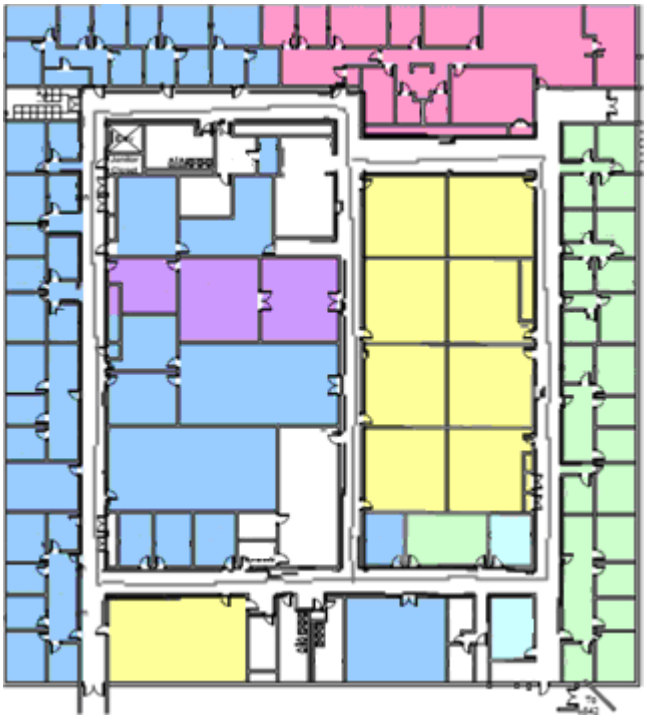


Figure 4.11: Overlaid map

Table 4.3: Statistics of ASRs in Figure 4.8 and Figure 4.9

	Master Robot		Virtual Robot	
ASR ID	Num of Polylines	Num of Vertices	Num of Polylines	Num of Vertices
1	5	15	10	38
2	9	37	15	59
3	18	70	15	76
4	25	103	18	84
5	12	53	16	80
6	14	67	10	48
7	16	63	12	53
8	21	81	18	72
9			22	98
10			15	64
11			19	76
12			14	51
13			15	57
Total	120	489	199	856
Avg	15	61	15	66

The memory space required for the particles of the original FastSLAM algorithm with global grids is 64/.25 x 64/.25 x 400=25 Mbytes to map the third floor of bldg with 400 particles. The memory space used for the particles of the implementation to map the same region is (40/.25 x 40/.25 + 1345 x16) x 400=18 Mbytes. Here, the value of 1345 is the total number of vertices in the global ASR and 16 represents the size of a vertex in bytes in the implementation. The memory space required for all the local grids is 9.766 Mbytes while the ASRs require 8.2 Mbytes. The memory space used for the simulation doesn't show much reduction from the one for the grid-based FastSLAM algorithm. However, when mapping area is larger the area mapped in the simulation, this memory constraint by local grids would be trivial compared to the total memory size it would require had the original mapping algorithm been used. Figure 4.12 is a regular grid map of the area of interest developed without ASR extraction method.

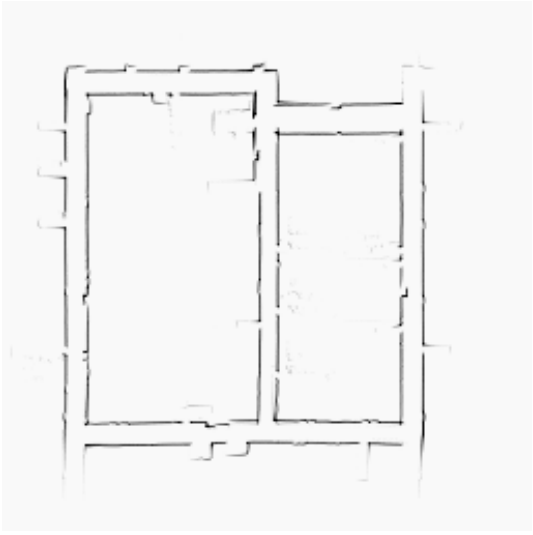


Figure 4.12: Grid map

The grid map includes every detail captured by the radar sensor. Although the map constructed from the ASRs miss details, it still shows all the main structures without sacrificing too much of details. Next, the ASR map generated by 400 particles is compared with the one developed by using 200 particles only to testify that the quality of a map improves as the number of particles increase in FastSLAM. Figure 4.13 displays a map produced by 200 particles.

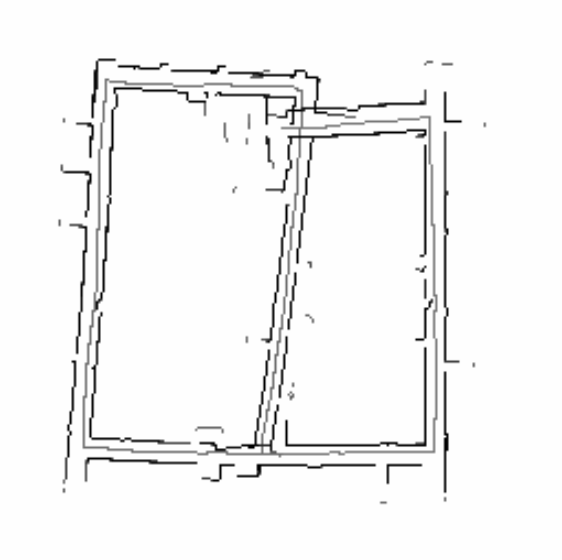


Figure 4.13: Global map of ASRs with 200 samples

As expected, the map doesn't represent the blueprint in Figure 4.1. It is far from truth. The last section of the chapter summarizes the testing results.

4.3 Testing Summary

The implementation of FastSLAM for multi-robots using local grids and ASRs as global map representation for the samples is satisfactory. The maps produced by the implementation represent the blueprints perfectly. The implementation has also showed that the quality of a map improves as the number of particles increases within FastSLAM. The map presented in Figure 4.10 could be produced by using local maps and ASR global map representation. Had I run a simulation for a map of the same area using regular, global grids, it would have been limited to using less than 200 particles on a personal computer. The time taken to produce the map in Figure 4.10 by the implementation is 4.9 seconds. This is a tolerable time lapse when the algorithm is implemented online because a robot would take longer than the time taken when it travels the path of Robot1 in Data1. The next chapter concludes the thesis and offers a future work prospect.

5. Future Work and Conclusion

The test results show that the implementation is successful for the purpose of the thesis. However, some of unexpected behaviors are noticed while in testing. Section 5.1 will encapsulate the implementation and draw a conclusion and section 5.2 will offer the direction of future work.

5.1 Testing Results Analysis

Although the fastSLAM algorithm is a powerful solution to a SLAM problem, current available implementations of the algorithm use only grid map representation and let each particle maintain a global occupancy grid map. This will certainly limit the capability of the algorithm. In the implementation, this limitation was moderately relieved by using Absolute Space Representations. Although only small improvements in memory usage were shown by the simulations, it would have been larger had the mapping areas been larger than the testing areas due to a local grid size being fixed for ASR global maps while a global grid size in a grid-based FastSLAM mapping grows.

In addition, when multi-robots are involved in mapping, the solution requires as many filters to be set up as the number of the participating robots. In this case, the number of particles that can be used in a filter would be significantly limited with the grid-based mapping of the FastSLAM algorithm. Thus, in a multi-robot FastSLAM mapping, it would be good choice to map using ASRs to alleviate the memory constraint

The objective of the thesis was to:

1. Implement and test the single-robot fastSLAM algorithm using ASRs. To achieve this task, the published probabilistic mapping method [69] was first modified to

let each particle possess a local grid not a global grid. Then local grids are converted into ASRs.

2. Implement and test a multi-robot FastSLAM. This was accomplished by letting a robot maintain a filter for each robot.

3. Develop a method to extract a polygonal map representation from a grid map and to simplify the extracted polylines. This task is accomplished by extracting polylines from the grid map according to the cell values. A neighboring cell with higher value is selected as the next cell to be inserted into a polyline.

All parts of the objectives were successfully implemented, and testing results show that in spite of local mapping, the performance of the algorithm is not reduced, rather improved the accuracy of the map by using more number of particles. Maps from a simulation with 400 particles showed better quality than the ones from a simulation with 200 particles.

Although the implementation is successful, I noticed a strange behavior of the implementation. Theoretically, as the number of particles used for a simulation is increased, the better map is supposed to be obtained. However, in a few of runs, output maps improved when the number of particles is increased from 200 to 400, but not when the number is changed from 400 to 450. This is against the nature of FastSLAM. There are only two possible explanations with the behavior: either the method of calculating the accurateness of a map when range data is incorporated into the map is not reliable or the random number generator is not working in a consistent manner. I think the latter explanation carries more weight.

5.2 Future Work

Although the implementation was satisfactory, it contains an inexplicable behavior as the number of particles increase. The strange behavior of the implementation is abnormal considering the nature of FastSLAM. One possible cause of this behavior is the weight calculation method used for the implementation. Further research is required to test whether this error calculation method is reliable. Another possible cause may lie with the random number generator. I would like to research the behavior of the generator more closely and find the reason of the abnormality of the implementation and correct the problem.

Appendix A. Algorithms

A.1 Robot Pose Estimation Method

```
Robot Pose Estimation Method:
  1. Initial sample pose= the intial robot pose if the
     pose is known, otherwise to (0,0,0)
  2. Odom_pose=Lidar-corrected pose=1st odemetry reading
  3. Calculate Delta=old odom_pose – new_odom_pose
  4. Per Particle i:
      4.1 sigma(i)=Action_model * delta
      4.2 Sample_pose(i)=delta + gaussian_random(sigma)
```

Figure A.1 Robot pose estimation method

A.2 Algorithm to Convert Vector Map to a Grid Map

```
ASRtoGrid (ASR)
  Loop(line)
     Loop(v1, v2)
        xlen=v2.x-v1.x
         ylen=v2.y-v1.y
         If(xlen>ylen & xlen>0) bin=xlen
         Else bin=ylen
         If (bin==0) dx=dy=0
         Else
            dx=xlen/bin
            dy=ylen/bin
            For(k=0 to k<bin)
                x=v1.x+k*dx
                y=v1.y+k*dy
                grid(x,y)=larger(value(v1),value(v2))
            end for
          end else
      end loop
   end loop
```

Figure A.2: Algorithm to convert vector map to a grid map

A.3 Algorithm to Etract Polylines

```
Extract polylines(grid)
  while
     PolyLine line
     v=grid(x_smallest, y_smallest)
     line.add(v)
     sg.unset(v)
     while
        v=getNextVertex(v)
        line.add(v)
        unset(v)
     ene while
     simplifyLine(line)
     addPolyLine(line)
  end while

Simplify Polyline (line)
   count=0
   lastlocation=0
   numVtx=line->length
   For (i=0 to i<numVtx)
      bgvtx=line[0]
      If(i==1)
         Lastlocation=1
         Currentvtx=line[1]
         Prevang=getlineangle(bgvtx, currentvtx)
          diff=prevang-currentang
         if |diff|<threshold
             line.remove(count+1)
         else
             count++
         end if
         Lastlocation=count+1
         Prevang=getlineangle(line[count], currentvtx)
      End if
   End for
```

Figure A.3: Algorithm to extract polylines

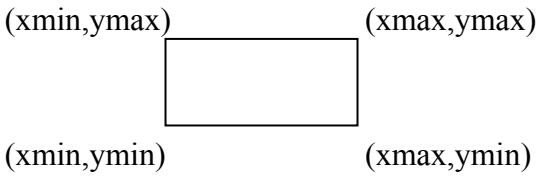
A.4 ASR Map

Each ASR contains

- ASR number
- Polylines
- xmin, xmax, ymin, ymax

Figure A.4: ASR

A window of an ASR is defined by the four corners as shown below.



The values stored in the four variables are global. “xmin” and “ymin” are the offsets to be added to a cell of a grid when the grid from an ASR is to be transferred into the global map.

A.5 Robot Pose Addition and Subtraction

```
Pose2_add(a,b)
  c.x=cos(b.θ)*a.x – sin(b.θ)*a.y +b.x
  c.y=sin(b.θ)*a.x – cos(b.θ)*a.y +b.y
  c.θ=atan2(sin(a.θ+b.θ), cos(a.θ+b.θ))

Pose2_sub(a,b)
  c.x=cos(b.θ)*(a.x-b.x)+sin(b.θ)*(a.y-b.y)
  c.y=-sin(b.θ)*(a.x-b.x)+cos(b.θ)*(a.y-b.y)
  c.θ=atan2(sin(a.θ-b.θ), cos(a.θ-b.θ))
```

Figure A.5: Robot pose addition and subtraction

Bibliography

1. J. Borenstein, B. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Wellesley, MA, 1996.

2. A.P. Dempster, A.N. Laird, and D.B. Rubin. *Maximum likelihood from incomplete data via the EM algorithm*. Journal of the Royal Statistical Society, Series B, 39(1):1–38, 1977.

3. J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. A practical, decision-theoretic approach to multi-robot mapping and exploration. In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2003.

4. B.J. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. Journal of Robotics and Autonomous Systems, 8:47–63, 1991.

5. D. Fox, W. Burgard, and S. Thrun. *Markov localization for mobile robots in dynamic environments.* Journal of Arti_cial Intelligence Research (JAIR), 11:391.427, 1999.

6. Kristopher R. Beevers. *Topological Mapping and Map Merging with Sensing-limited Robots.* Master's thesis, Rensselaer Polytechnic Institute, 2004.

7. J. Borenstein, B. Everett, L. Feng, *Navigating Mobile Robots: Systems and Techniques,* A.K. Peters, Wellesley, MA, 1996.

8. W. Burgard, A. Derr, D. Fox, A.B. Cremers, *Integrating global position estimation and position tracking for mobile robots: The dynamic markov localization approach*, in: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98), Victoria, BC, 1998.

9. W. Burgard, D. Fox, D. Hennig, T. Schmidt, *Estimating the absolute position of amobile robot using position probability grids*, in: Proc. AAAI-96, Portland, OR, 1996.

10. I.J. Cox, J.J. Leonard, Modeling a dynamic environment using a Bayesian multiple hypothesis approach, Artificial Intelligence 66 (1994) 311–344.

11. S. Thrun,W. Burgard, and D. Fox, *A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping,* In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA, 2000. IEEE.

12. J.-S. Gutmann, C. Schlegel, *Amos: Comparison of scan matching approaches for self-localization in indoor environments*, in: Proc. 1st Euromicro Workshop on Advanced Mobile Robots, IEEE Computer Society Press, 1996.

13. Jensfelt, S. Kristensen, *Active global localisation for a mobile robot using multiple hypothesis tracking*, in: Proc. IJCAI Workshop on Reasoning with Uncertainty in Robot Navigation, Stockholm, Sweden, 1999, pp. 13–22.

14. F. Lu, E.Milios, Globally consistent range scan alignment for environment mapping, Autonomous Robots 4 (1997) 333–349.

15. S. Thrun, Bayesian landmark learning for mobile robot localization, Machine Learning 33 (1) (1998).

16. Simmons, R. and Koenig, S. 1995. *Probabilistic robot navigation in partially observable environments*. In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI).

17. King, S. and Weiman, C. 1990. Helpmate autonomous mobile robot
navigation system. In Proc. of the SPIE Conference on Mobile Robots, Vol. 2352, Boston, MA, pp. 190–198.

18. Wolfart, E., Fisher, R.B., and Walker, A. 1995. *Position refinement for a navigating robot using motion information based on honey bee strategies.* In Proc. of the International Symposium on Robotic Systems (SIR 95), Pisa, Italy.

19. Kurazume, R. and Shigemi, N. 1994. *Cooperative positioning with multiple robots.* In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

20. Rekleitis, I.M., Dudek, G., and Milios, E. *Multi-robot exploration of an unknownenvironment, efficiently reducing the odometry error.* In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1340-1345. 1997.

21. Konolige, K. 1999. *Markov localization using correlation*. In Proc. Of the International Joint Conference on Artificial Intelligence (IJCAI). (113) 1999.

22. Vidakovic, B. "Sequential Monte Carlo Methods," IsyE8843A, Handout 13, 5 Feburary 2007 http://www2.isye.gatech.edu/~brani/isyebayes/bank/handout13.pdf

23. Thrun, S. & Bücken, A.Learning Maps for Indoor Mobile Robot Navigation *School of Computer Science, Carnegie Mellon University,* 1996

24. Joss Knight, Andrew Davison and Ian Reid. *Towards constant time SLAM using postponement.* Proceedings. 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001**.**

25. Oscar Serrano Serrano. *Robot Localization using wireless networks.* Informe Tecnico en elaboracion Departamento de Informatica, Estadısticay Telematica, 2003

26. Oscar Serrano, Jose Maria Canas, and Vicente Matellan, *Robot Localization using Wifi Signal without Intensity Map*. Proceedings of the V Workshop Agentes Fisicos WAF, 2004, 4, 79-88

27. Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. *A probabilistic approach to collaborative multi-robot localization*. In Special issue of Autonomous Robots on Heterogeneous Multi-Robot Systems, 8(3), 2000.

28. A.M. Jazwinsky. *Stochastic Processes and Filtering Theory*. Academic Press, New York, 1970.

29. Jose Guivant, Eduardo Nebot. *Optimization of the simultaneous localization and map building algorithm for real time implementation*. IEEE Transactions on Robotics and Automations, 2001. 17, 242-257.

30. S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A.Y. Ng. *Simultaneous mapping and localization with sparse extended information filters.* In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, Proceedings of the Fifth International Workshop on Algorithmic Foundations of Robotics, Nice, France, 2002.

31. Kristopher R. Beevers. Topological Mapping and Map Merging with Sensing-limited Robots, PhD. Dissertation, Rensselaer Polytechnic Institute, 2004.

32. B.J. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. Journal of Robotics and Autonomous Systems, 8:47–63, 1991.

33. D. Pagac, E.M. Nebot, and H. Durrant-Whyte. *An evidential approach to map-building for autonomous vehicles*. IEEE Transactions on Robotics & Automation, 14(4):623–629, August 1998.

34. P.E. Rybski, F. Zacharias, J.-F. Lett, O. Masoud, M. Gini, and N. Papanikolopoulos. *Using visual features to build topological maps of indoor environments.* In Proceedings of the 2003 IEEE International Conference on Robotics & Automation, Taipei, Taiwan, September 2003.

35. H. Shatkay and L.P. Kaelbling. *Learning topological maps with weak local odometric information.* In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 1997.

36. S. Thrun and A. Bucken. *Integrating grid-based and topological maps for mobile robot navigation.* In Proceedings of the Thirteenth National Conference on Artificial Intelligence, pages 944–950, 1996.

37. S. Thrun, J. Gutmann, D. Fox, W. Burgard, and B. Kuipers. *Integrating topological and metric maps for mobile robot navigation: A statistical approach*. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1998.

38. Sebastian Thrun. *Robotic mapping: A survey*. Technical Report CMU-CS-02-111, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 2002.

39. N. Tomatis, I. Nourbakhsh, and R. Siegwart. *Combining topological and metric: a natural integration for simultaneous localization and map building.* In Proceedings of the Fourth European Workshop on Advanced Mobile Robots (Eurobot 2001), pages 19–21, Lund, Sweden, September 2001.

40. N. Tomatis, I. Nourbakhsh, and R. Siegwart. *Hybrid simultaneous localization and map building: closing the loop with multi-hypothesis tracking.* In Proceedings of the 2002 IEEE International Conference on Robotics & Automation, *3*, pp. 2749-2754, May 2002.

41. M11: B. Tovar, S. LaValle, and R. Murrieta. *Optimal navigation and object finding without geometric maps or localization.* In Proceedings of the 2003 IEEE International Conference on Robotics & Automation, Taipei, Taiwan, September 2003.

42. Wikipedia, 14 January 2007
http://en.wikipedia.org/wiki/Expectation-maximization_algorithm.

43. J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. *A practical, decision-theoretic approach to multi-robot mapping and exploration.* In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2003.

44. Dieter Fox. *Distributed multi-robot exploration and mapping*. In Proc. of the 2nd Canadian Conference on Computer and Robot Vision (CRV'05), 2005-Volume 00.

45. Laura E. Barnes, Richard Garcia, Todd M. Quasny, and Larry D. Pyeatt. *Multi-agent mapping using dynamic allocation utilizing a centralized storage system.* 12th Annual Mediterranean Conference on Control and Automation, 2004

46. Juan Andrade-Cetto. *Multirobot C-SLAM: Simultaneous localization, control and mapping*. In Proc. IEEE Int. Conf. Robot. Automat. pp 324-329, Barcelona, Apr. 2005.

47. Andrew Howard, Gaurav S. Sukhatme, and Maja J. Mataric. *Multi-robot mapping using manifold representations.* IEEE International conference on robotics and automation, New Orleans, Louisiana, Apr 2004, pages 4198-4203.

48. Kurt Konolige, Dieter Fox, Benson Limketkai, Jonathan Ko, Benjamin Stewart. *Map merging for distributed robot navigation*, In Proc. of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems.

49. Haye Lau. *Behavioural Approach for Multi-Robot Exploration*. Proceedings of. 2003 Australasian Conference on Robotics and Automation, Brisbane, Australia, Dec, 2003.

50. Andrew Howard. Multi-robot Simultaneous Localization and Mapping using Particle Filters. Proceedings of Robotics: Science and Systems, 2005.

51. W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. *Collaborative multi-robot exploration.* In Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2000.

52. R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. *Coordination for multi-robot exploration and mapping*. In Proc. of the National Conference on Arti_cial Intelligence (AAAI), 2000.

53. Andrew J. Davison and David W. Murray. *Simultaneous Localization and Map-Building Using Active Vision*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24, 865-880.

54. Stefan B. Williams, Gamini Dissanayake, Hugh Durrant-Why. *An efficient Approach to the Simultaneous Localisation and Mapping Problem*. Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on, 2002, 1.

55. S.B. Williams, G. Dissanayake, and H. Durrant-Whyte. *Towards multi-vehicle simultaneous localisation and mapping.* In Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2002.

56. S. Thrun and Y. Liu. *Multi-robot SLAM with Sparse Extended Information Filters.* In Proc. of ISRR, Sienna, Italy, pages 322-331. ACM 2003.

57. J.W. Fenwick, P.M. Newman, and J.J. Leonard. *Cooperative concurrent mapping and localization.* In Proc. of the IEEE International Conference on Robotics & Automation (ICRA), 2002.

58. M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. *FastSLAM: A factored solution to the simultaneous localization and mapping problem*. In Proc. of the National Conference on Arti_cial Intelligence (AAAI), 2002.

59. K. Murphy. *Bayesian map learning in dynamic environments*. In Advances in Neural Information Processing Systems (NIPS), 1999.

60. Matthew Walter and John Leonard. *An experimental investigation of cooperative SLAM.* Proceedings of the Fifth IFAC/EURON Symposium on Intelligent Autonomous Vehicles,(Lisbon, Portugal), July, 2004.

61. A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo in Practice.* Springer-Verlag, New York, 2001.

62. B. Stewart, J. Ko, D. Fox, and K. Konolige. *A hierarchical bayesian approach to mobile robot map structure estimation.* In Proceedings of the Conference on Uncertainty in AI (UAI), Acapulco, Mexico, 2003.

63. Wenyan Hu, Tom Downs, Gordon Wyeth, Michael Milford and David Prasser. *A Modified Particle Filter for Simultaneous Robot Localization and Landmark Tracking in an Indoor Environment*. Proceedings of the Australasian Conference on Robotics & Automation, 2004.

64. Michael Montemerlo, Sebastian Thrun. *Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM*. Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on, 2003*, 2.*

65. Austin Eliazar and Ronald Parr. *DP-SLAM: Fast, Robust simultaneous and Mapping Without Predetermined Landmarks.* Proc. of the International Joint Conference on Artificial Intelligence (IJCAI), 2003*,* 1135-1142.

66. M.A. Paskin. *Thin junction tree filters for simultaneous localization and mapping*. Technical Report UCB/CSD-02-1198, University of California, Berkeley, CA, 2002.

67. Kennard R. Laviers. A Thesis Submitted to the Graduate Faculty of Air Force Institute of Technology for the Degree of MASTER OF SCIENCE, October, 2004.

68. K. Konolige, J.-S. Gutmann, D. Guzzoni, R. Ficklin, and K. Nicewarner. *A mobile robot sense net.* In Proceedings of SPIE 3839 Sensor Fusion and Decentralized Control in Robotic Systmes II, Boston, September 1999.

69. *SourceFORGE.net*. 7 October 2006 http://playerstage.sourceforge.net/

70. J. Theiler and B. Gaugh, Discrete.c, 8 January 2007
http://www.koders.com/c/fid9477B49B46EB00F14E586C1B85490FCFCF6AC088.aspx?s=sort

Vita

Choyong G. Koperski was born in the city of Iri, South Korea, in 1967. She received a Bachelor's degree in computer engineering from University of Nebra, Lincoln, Nebraska, in 2002. She is currently a M.S. student at the Air Force Institute of Technology, where she is researching multi-robot SLAM with cognitive applications under the supervision of Professor Gilbert Peterson.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
22-03-2007	Master's Thesis	Aug 2005 – March 2007

4. TITLE AND SUBTITLE

Multi-Robot FastSLAM for Large Domains

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Koperski, Choyong G., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/07-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Mikel M. Miller (937) 255-6127 x4274
AFRL/SNRN
2241 Avionics Circle
WPAFB, OH 45433-7765
mikel.miller@wpafb.af.mil

10.SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This thesis presents a mapping algorithm that extends single-robot FastSLAM algorithm to a multi-robot mapping algorithm. FastSLAM, which is based on the Rao-Blackwellized particle filter, solves the SLAM problem for single-robot mapping using particles to represent the posterior estimate of the robot pose and the map. Each particle of the filter possesses its own global map which is likely to be a grid map. The memory space required for these entire maps pose a serious limitation to the algorithm's capability when the problem space is large. This problem will only get worse if the algorithm is adapted to a multi-robot mapping. In the proposed mapping algorithm, each particle of the filter still maintains a local grid to map its vicinity and periodically this grid map is converted into an Absolute Space Representations. An Absolute Space Representation expresses a world in polygons requiring only minimal amount of memory space. By using this altered mapping strategy, the problem faced in FastSLAM when mapping a large domain can be alleviated.

15. SUBJECT TERMS
SLAM, multi-robot SLAM, Absolute Space Representations, ASR, particle filter, FastSLAM

16. SECURITY CLASSIFICATION OF:			17.LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b.ABSTRACT	c.THIS PAGE			Gilbert Peterson, Dr., USAF
U	U	U	UU	104	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, ext 4281 gilbert.perterson@afit.edu